A DECISION SUPPORT SYSTEM FOR DYNAMIC SCHEDULING: PRACTICE AND THEORY

By

HALDUN AYTUG

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1993

ACKNOWLEDGEMENTS

I would like to thank my advisor Professor Gary J. Koehler for his constant support and assistance during this dissertation, and for his help during moments of frustration. I am thankful to the other members of my committee, Professor Chung Yee Lee, Professor Anthal Majthay, and Professor Richard A. Elnicki, for their input. I would like to thank Professor Selcuk Erenguc for his guidance as a graduate coordinator during my studies at the University of Florida. I am grateful to friends who helped make life enjoyable during my stay in Gainesville. Finally, I am most grateful to my family for their constant love and encouragement.

Last thanks go to International Business Machines Corporation for their support on part of this dissertation.

TABLE OF CONTENTS

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

A DECISION SUPPORT SYSTEM FOR DYNAMIC SCHEDULING:
PRACTICE AND THEORY

By

Haldun Aytug

August 1993

Chairman: Professor Gary J. Koehler
Major Department: Decision and Information Sciences

Knowledge acquisition is a major bottleneck in successfully implementing an Expert System in a production environment. Moreover, it is very questionable that a domain expert exists in this domain, due to the dynamic and complex nature of the environment. Knowledge base update is another problem with the Expert System approaches for the scheduling domain since some production environments are subject to frequent changes making the existing knowledge base obsolete. A remedy to the knowledge acquisition and update problems is using "Machine Learning" approaches. This reduces the dependence on a human expert and automates the knowledge acquisition and update processes.

In this research we motivate the need for using machine learning techniques for artificial intelligence based production scheduling systems and demonstrate how a classifier system/genetic algorithm hybrid approach can be used to generate an acceptable

knowledge base. Due to their continuous feedback control mechanisms, classifier systems are very appropriate for updating knowledge bases when the environment is changing. We have developed a learning system that is integrated with a simulation environment (developed in C++) which supports inference. A rule language supportive of the genetic search operators used by the system has been developed. Two subsystems the "learning subsystem" and the "task subsystem" interact throughout the "knowledge construction" process. The learning subsystem evaluates the performance of the task subsystem after every learning episode and generates an expectedly better knowledge base for the task subsystem to use in the next learning episode. This procedure stops after a reasonable knowledge base is constructed.

The theoretical part of this research involves developing a worst case bound on the number of learning episodes the system has to perform before stopping with a well performing knowledge base. This is equivalent to developing a stopping criteria for genetic algorithms since classifier systems use genetic algorithms to search for new rules. To date there has been no series attempt to develop a stopping criteria for Genetic Algorithms. We have developed an upper bound on the number of iterations a GA has to execute to assure seeing an optimal solution with a specified confidence level by using the concept of first passage times of a Markov Chain. For a given string length, we have also found the optimal number of iterations necessary before stopping.

CHAPTER 1
INTRODUCTION

Scheduling has long been an interest of researchers both from a practical and a theoretical point of view. Scheduling is a problem encountered in every type of business. Examples are project scheduling, flight scheduling, scheduling classes at a university, production scheduling, etc. Whatever the application is, scheduling involves assigning scarce resources to operations. In this dissertation we will restrict our attention to production scheduling problems.

In a production environment, schedules take the form of assigning operations to resources such as machines, and labor in a sequence that will optimize some performance criteria.

It is possible to define the scheduling problem at different levels. At the highest level scheduling is more of a long term deployment of resources to production types at an aggregate level, whereas at the lowest level, scheduling involves the actual assignment of each individual resource for each individual operation. In this study we will focus on the lowest level, the actual allocation of resources to operations.

There has been an extensive amount of work in this area. Traditional optimization techniques offer high quality and elegant solutions to a very restricted subset of these problems. Due to their restrictive nature, models solved this way have very limited generalizability.

Recently, there has been a trend towards increasing the generalizability of the scheduling models. These approaches can model very complex environments, but due to the complexity of the problem they offer lower quality solutions with a very high degree of applicability. Artificial Intelligence (AI) techniques have been very promising for solving real time scheduling problems. This can be attributed to their ability to incorporate domain specific knowledge into the search process and their flexibility in modeling different aspects of the scheduling environment.

In the following sections we will briefly introduce different approaches to the scheduling problem.

1.1 Scheduling and Operations Research

Scheduling as a sequencing problem has attracted much attention from the Operations Research community. Starting with Johnson's work on flow shop problems (Baker, 1974), and Jackson's on single machine problems (Lawler et al., 1985) in 1954, research has progressed to more challenging problems. As the problem types evolved to more complex forms and became more representative of the scheduling environment, the algorithms developed by researchers started to fail to achieve the optimal solutions within reasonable time frames. It was not until the foundation of the complexity theory that researchers realized they were dealing with inherently very hard problems. Starting with this new knowledge, research on sequencing shifted to developing heuristic algorithms that find approximate solutions for these hard problems.

Most problems tackled by optimization methods consider very simplistic environments. Typical problems attacked involve a few machines, infinite resources and jobs with deterministic attributes. The stochastic nature of the problem is usually completely ignored.

Performance measures considered are not any more realistic than the model assumptions. Usually simple criteria such as flow time, lateness, etc. have been used to measure the performance of the algorithms developed. These measures have been selected since they are functions that are relatively easy to work with. However, there has been almost no effort to investigate cost or quality related measures.

There has been some work on the stochastic scheduling problems, but due to their complexity, analytical models involve very restrictive assumptions about probability distributions.

When complete solution methods do not exist, it has been common practice to apply rules of thumb to solve the problem. Dispatching rules are simple heuristics that are found to work well in stochastic environments.

Simulation has almost never been used for optimization on scheduling problems due to the fact that the approach itself does not suggest a way of solving the problem. It has almost exclusively been used to test how good an algorithm performs when certain assumptions of the algorithm are relaxed. Simulation has been extensively utilized to compare the performances of heuristic rules in non-deterministic environments.

In a relatively recent survey by Lageweg et al. (1981), it has been reported that problems that are optimally solvable are less than ten percent of the total number of problems tackled by optimization methods.

Even though thirty years of research has compiled a large database of algorithms for scheduling problems. Most of these are not directly applicable for real time control of scheduling operations. It is only during the last decade that the researchers have focused on methods that can connect this background knowledge to real environments.

1.2 Artificial Intelligence and Scheduling

During the last twenty years there have been enormous improvements in computing technology. Current personal computers are almost as powerful as yesterday's mainframes and are easily accessible. Most production environments are computer integrated. Applications of computing are more ubiquitous. Indeed, most production environments are now computerized.

An early endeavor of computer scientists was to endow computers with the ability to perform intelligently. This area of research is called Artificial Intelligence (AI). The tremendous advances in computers have made AI a more tractable area. Today it is possible to write software that outperforms humans.

AI techniques have been successfully implemented in many domains. For example, Expert Systems (ESs) have found large application domains from banking to medical diagnosis and are very effective for solving such problems. Pattern recognition and natural language packages are ready to be commercialized.

During the last decade there has been an enormous number of AI applications on scheduling. Researchers developed ESs for many different production environments, (Alpar and Srikanth 1989a), (Chang 1985), (Kerr and Ebsary, 1988), (O'Keefe 1986). It was discovered that knowledge base construction was the main bottleneck to building such systems. ESs were not easy to maintain, and their simple inference mechanisms proved to be insufficient for most problems. Yet ESs were able to solve a lot of problems and help achieve some of the management objectives (Miller, Lufg and Walker 1988).

Other AI applications focus on search heuristics and knowledge representation. Knowledge representation is important because the system should be able to represent domain specific knowledge. Search heuristics are important because most AI applications focus on real time scheduling where time is a critical constraint. Search heuristics, sometimes referred to as meta knowledge, should efficiently search for the solution and must exploit knowledge representation language characteristics. Search schemes such as Constrained Heuristic Search, (CHS) (Arff and Hasle 1990), (Fox, 1983), (Fox and Sycara, 1990), blackboard style control, (Ow, Smith and Thriez 1988), (Smith, Ow and Potvin, 1990), planning algorithms (Shaw, 1988c) have been successfully applied.

Even though these systems are able to successfully utilize AI techniques, they lack one important aspect of intelligence: "learning." There has been little work on developing scheduling systems that can learn. Even though the learning problem has been analyzed in generic settings, their results are not directly applicable to the scheduling domain.

Recently, there has been work on heuristic discovery using learning algorithms. Such systems use a learning algorithm to generate a knowledge base and utilize this knowledge for problem solving (Piramuthu et al., 1991), (Yih 1990).

Due to their ability to represent complex domains and apply domain specific knowledge, AI approaches are most promising for scheduling. They not only provide powerful problem solving mechanisms but also blend well with today's computer integrated manufacturing environment.

1.3 Scheduling and Genetic Algorithms

Genetic Algorithms (GAs) are general purpose, stochastic search algorithms. They conduct the search by manipulating bit strings and employ stochastic operators such as reproduction, crossover and mutation. At each iteration a GA moves from one population of strings to another until a set of useful strings are found.

Until recently, little work has been done on the search behavior of GAs. Holland's schemata theory is of little help to explain the convergence behavior of GAs. Recently there has been significant work on analyzing GAs as Markov Chains (Nix and Vose, 1992). Results are still far from applicability but the model introduces powerful analytical tools for investigating GA search characteristics and worst case analyses.

Genetic operators have been employed for sequencing problems. There has been research on developing new operators that are suitable for sequencing problems (Whitley, Starkweather and Fuquay, 1989), (Cleveland and Smith, 1989).

Classifier Systems (CSs) are adaptive learning systems that use GAs as rule discovery algorithms. Their application to scheduling has been limited due to their weak knowledge representation language. Nevertheless, it has been suggested that they are robust learning systems in dynamic domains (Goldberg 1990). First attempts to use them to learn sequencing heuristics rather than sequences have proven promising (Hilliard et al., 1987), (Hilliard et al., 1989).

1.4 Contents of This Dissertation

Most optimization approaches to scheduling suffer from their overly simplistic assumptions, their inability to incorporate domain specific knowledge and their inflexibility in adjusting to environment changes. Even though such techniques are capable of producing high quality solutions with respect to their presumed environment, solutions generated by such methods are not directly applicable for real time scheduling. The solutions generated may even be infeasible because of their overly simplifying assumptions. Among all OR techniques, only simulation offers adequate representational power. It is also very useful in performing "what if" analysis.

Almost all AI approaches employ enough expressive power for representing domain characteristics. Expert Systems lack the adequate search power necessary to search the rule space efficiently enough. Among the AI approaches to the scheduling problem, most of the recent work has been towards using domain specific heuristic algorithms, together with an appropriate knowledge representation scheme. Even though

these methods are successful in solving specific problems, they are unable to adapt their search mechanisms as the environment changes. They lack the ability to learn.

Applications of Machine Learning (ML) in scheduling problems are at an early stage. Even though some encouraging results exist, pure ML approaches are too general for a complex domain like scheduling. In this dissertation we concentrate on a hybrid approach which combines the knowledge generated by OR research over the last thirty years and the knowledge representation techniques enabled by AI research. A GA search mechanism is employed to discover cases where certain heuristics are applicable. We test the plausibility of a GA based learning approach. We focus our attention on a flowshop environment which is complex enough to test the usefulness of such an approach. An object oriented simulation environment that is capable of handling inference is utilized.

We also study the behavior of GAs as search algorithms. We present a bound on the computational complexity of GAs. We prove some properties of this bound and demonstrate how optimal GA search parameters can be obtained to minimize the computational complexity of GA under certain conditions.

In Chapter 2, we present a literature survey of OR and AI approaches to scheduling. Emphasis is on dispatching rules and AI applications in scheduling. In this chapter we discuss limitations and advantages of both AI and OR approaches. We also motivate the need for learning in real time scheduling systems.

Chapter 3 mainly focuses on learning. Part of the chapter is dedicated to Machine Learning (ML) research and some theoretical issues involved. We also summarize the

results of some earlier work on ML. Finally, we analyze the existing research on scheduling from a ML perspective.

We review the literature on GAs and CSs in Chapter 4. We start with earlier theoretical findings on GAs. Next, we present a discussion of several applications of GAs and CSs in other domains as well as in the scheduling domain. A large part of this chapter will be devoted to the results of Vose and Liepins, (1991), and Nix and Vose, (1992) which will provide building blocks for Chapter 5.

In Chapter 5, we develop a bound on run time complexity of a GA. We derive the bound through a series of theorems and lemmas. Part of the analytical results we need are borrowed from the research on spectral analysis of non-negative matrices.

Chapter 6 summarizes our first attempt to build a learning system using GAs. We first analyze a simple, hypothetical production environment and compare the performance of the learning system against systems using single heuristics with no learning. We further investigate the possibility of learning optimal rules by simulating an environment where an optimal rule is known to exist.

Chapter 7 extends the idea described in Chapter 6 to a flowshop environment. A rule language that is able to capture the knowledge related to a flowshop environment is described. Results of simulation experiments are presented and discussed by comparing to some known dispatching heuristics.

Finally, in Chapter 8, conclusions and directions for future research for both practical and theoretical problems is discussed.

CHAPTER 2

SCHEDULING, DISPATCHING AND ARTIFICIAL INTELLIGENCE

2.1 Overview of Scheduling and Operations Research

Research on scheduling problems has a long history, dating back to the late nineteen fifties and early nineteen sixties. Most of the optimal rules and algorithms for certain classes of problems were found during these initial stages. As production systems evolved and became more complex, mathematical models representing these real systems have become more complex. After Stephen Cook introduced the theory of NP-completeness in 1971 (Gary and Johnson 1979), researchers realized that most of the scheduling problems belonged to the class of computationally intractable problems, namely NP-complete problems. This triggered research on heuristic algorithms for scheduling that do not guarantee optimal solutions but do guarantee a good solution within some error limits of the optimal solution. It was also shown that these models could deal with comparatively realistic problem types and appeared to be more robust than optimization based methods (Rodammer and White 1988). Recent results have shown that even these heuristic algorithms have their limitations. It was believed that for any NP-complete problem and given error tolerance there existed an approximation algorithm that could solve the problem within the specified error bounds, but it was shown that for certain class of problems even approximation is NP-complete.

In this section we review production scheduling problems and discuss the advantages and disadvantages of traditional Operations Research (OR) methods used for their solution.

2.1.1 Definition of the Scheduling Problem and the Production Environment

From an OR point of view, the scheduling problem can be defined as finding a sequence of operations on a given set of jobs that will give an optimal solution with respect to the prespecified performance criterion. There may be more than one measure of interest for a given problem formulation. Each may result in different solutions. Also, a slight change in the problem may drastically change its mathematical modelling. It is often the case that there may exist a polynomial time algorithm for a model but a slight change may yield a problem without exact solution methods in polynomial time.

A production environment is defined by the number of machines, the type of machines, and the nature of the flow on the shop floor. All queues feeding into the machines are assumed to have infinite buffer sizes. All machines are assumed to be able to process only one operation at one time.

When a production environment has a work flow that is unidirectional where each job visits each machine installed in a series, we refer to it as a flowshop. On the contrary, if the jobs have routes specified a priori, but the routes do not enforce a unidirectional flow, such an environment is called a job-shop.

Jobs are usually defined by three attributes: processing times, release times and due dates. Processing times are assumed to be deterministic, i.e., the actual length of the

operation is known before the operation is preformed. The release time of a job is the time the job enters the shop floor and is available for processing. These are usually assumed to be zero or, if not zero, are assumed to be known a priori. It is implicitly assumed that the scheduler has perfect information as to when the job is going to be available. Due dates are the times when the jobs are supposed to leave the shop floor and be ready for shipment to the customer.

Machines are assumed to be the only type of scarce resource. Labor and raw materials are assumed to be available in infinite amounts. In some production environments with different types of job families, there may be sequence dependent setup times between two jobs coming from different families. There may also be technological constraints imposing precedence relationships between operations of a job. As an example, assume a job has five operations performed on it before being completed, and operation two has to be performed before operation five. This is referred to as a precedence relationship. In a flowshop environment such a relationship is implicit.

2.1.2 Notation and Parametric Description of Scheduling Problems

Let,

m be the number of machines;

n be the number of jobs;

p_{ij} be the processing time of job j on machine i;

p_j be the processing time of job j, (for one machine problems);

d_j be the due date of job j;

r_j be the release time of job j;

C_j be the completion time of job j;

F_j be the flowtime of job j;

L_j be the lateness of job j; and

T_j be the tardiness of job j.

The completion time is the time when a job is ready to leave the system. Flowtime, lateness and tardiness are defined as non-decreasing functions of the completion time as follows:

$$F_j = C_j - r_j,$$

$$L_j = C_j - d_j,$$

and

$$T_j = \max(0, L_j).$$

A function of completion time is called a "regular measure" with respect to the completion time if it is non-decreasing in completion time. Hence flowtime, lateness and tardiness are regular measures with respect to completion time.

Scheduling problems are defined by three parameters $\alpha/\beta/\gamma$ (Lawler et al. 1985).

The first parameter, α, is the description of the shop floor and can be any of the following:

1: The shop floor consists of only one machine.

mP: The shop floor consists of m parallel machines with identical speeds.

mF: The shop floor is a flowshop with m machines in sequence.

mJ: The shop floor is a job-shop with m machines.

The second parameter, β, is the description of the constraints or the attributes of the problem. If β is null then the problem has no constraints and the attributes are in their most general form. Some typical specifications are the following.

r_j: Release time of job (which is not necessarily zero.)

prec: There are precedence relationships between the jobs.

pmtn: Preemption is allowed.

$s_{i,j}$: Setup time of $s_{i,j}$ is required if job j is processed after job i when $j \neq i$. There is no setup required if j=i.

$p_j = p$: All processing times are equal to p.

The third parameter, γ, is the description of the objective function to be minimized. Usually these functions are regular measures with respect to the completion time of the job. Some typical performance measures are the following.

C_{max}: Maximum completion time, often referred to as "makespan".

$\sum_{i=1}^{n} C_i$: Total completion time.

F_{max}: Maximum flowtime.

$\sum_{i=1}^{n} F_i$: Total flowtime.

L_{max}: Maximum lateness.

$\sum_{i=1}^{n} L_i$: Total lateness.

T_{max}: Maximum tardiness.

$\sum_{i=1}^{n} T_i$: Total tardiness.

$\sum_{i=1}^{n} w_i T_i$: Total weighted tardiness.

In an excellent survey by Lawler et al., (1985) many scheduling problems are listed and solution methods are discussed. Similarly Lageweg et al., (1981) present complexity results for 4,536 problems and some references. It is not surprising to see that only 9% of these problems are listed as easy problems with 81% as NP-complete. The remaining 10% are open problems. Some of these problems and their solution methods, if any, are given in Table 2-1.

Table 2-1 clearly demonstrates that, other than single machine problems with simple performance measures, scheduling problems are mostly intractable by classical OR techniques.

Most of the OR approaches look at oversimplified versions of the real problem, such as assuming deterministic processing times, known release times and well behaved quantifiable objective functions. Even with such oversimplifying assumptions most of these problems fall into the class of NP-complete problems. Furthermore, solution methods suggested can not incorporate qualitative management objectives and heuristic knowledge into the problem.

In the next section, we present a picture of the "day to day" scheduling problem that practitioners face.

Table 2-1. Some scheduling problems

PROBLEM	COMPLEXITY	ALGORITHM
$1/prec/f_{max}$	P	Lawler, (Lawler et al. 1985).
$1/pmtn,r_j,prec/f_{max}$	P	Baker,Lawler,Lenstra and Rinnooy Kan, (Lawler et. al 1985).
$1//L_{max},T_{max}$	P	Earliest Due Date, (Baker 1974).
$1/r_j/L_{max}$	NP	Special cases are solvable by extended Jackson's algorithm, (Lawler et al. 1985)
$1//\Sigma C_j$	P	Shortest Processing Time, (Baker 1974)
$1//\Sigma w_j C_j$	P	Weighted SPT, (Baker 1974).
$1/r_j/\Sigma C_j$	NP	Pseudopolynomial algorithm by Lawler, (Lawler et al. 1985).
$1//\Sigma T_j$	NP	Generalized SPT, (Lawler et al. 1985).
$mP//\Sigma C_j$	P	(Lawler et al. 1985).
$mP//\Sigma T_j$	NP	(Lawler et al. 1985).
$mP//C_{max}$	NP	There are approximation algorithms, (Lawler et al. 1985).
$2F//C_{max}$, $2/pmtn/C_{max}$	P	Johnson's algorithm, (Baker 1974).
$2F/r_j/C_{max}$	NP	(Lawler et al. 1985).
$2F//L_{max}$	NP	(Lawler et al. 1985).
$2F//\Sigma C_j$	NP	(Lawler et al. 1985).
$3F//C_{max}$	NP	(Lawler et al. 1985).
$3F/pmtn/C_{max}$	NP	(Lawler et al. 1985).
$3F/pmtn/\Sigma C_j$	NP	(Lawler et al. 1985).

2.1.3 Scheduling Problem as it Appears in Industry

Even the simplistic mathematical models of the scheduling environment are hard, but as may be expected, real life problems are even harder. Smith explains the reasons as follows:

> The sources of difficulty identified included (1) the need to adhere to a typically idiosyncratic set of restrictions relating to production processes, resource capabilities, and resource availability, (2) the need to balance a large and conflicting set of objectives and preferences, and (3) the unpredictability of factory operation. (Smith 1988b, p. 382)

To make life more complicated, the performance measures are not usually as simple as those listed earlier. Management usually sets multiple objectives which not only interact but also conflict with each other. Sadeh and Fox explain this very clearly.

> Real-life scheduling problems are subject to a variety of preferences such as meeting due dates, reducing the number of machine set-ups, reducing inventory costs, using accurate and/or fast machines, making sure that some jobs are performed within a single work-shift, etc. Although these preferences are usually set independently to one another, they interact. For instance selection of a good start time for an activity (e.g. to meet a due date) may prevent the selection of an accurate machine for another operation or may prevent meeting another job's due date. For this reason, selecting operation start times or allocating resources based solely on local a priori preferences is likely to result in poor schedules. Preference propagation is meant to allow for the construction of measures that reflect preference interactions. These measures can then serve to guide the construction of a good overall schedule rather than a schedule that locally optimizes a subset of preferences. (Sadeh and Fox 1989, p. 1)

What happens in a real time situation is even more drastic than explained above.

> ...
> The operator responsible for all this has other worries, too. The tools the operator is responsible can't all run the same kind of products, and because the operator has a mixed stream of products, thought must be given to which product is going where. Furthermore, tools are idiosyncratic about when and how they age, and what effects that produces. Meanwhile, of course, he has to remember the process specifications and sequence of

> fabrication steps. Each wafer must go through, say, a photo process 10 or 15 times, each process building a layer resulting in a three dimensional complexity--you can't build level B before you build level A--. This complexity is compounded by other difficulties such as all boxes which store a lot of wafers look alike and often an important customer comes in with a special request. *Modern Times* at its worst. For operator and manager alike, the whole manufacturing process is like a giant chess game --except the complicated sequencing, combinatorial complexity, and unpredictability make it harder to think through than even the most demanding of games. (Sullivan and Fordyce 1990, p. 53[1])

Kerr and Ebsary state the reasons for choosing heuristic approaches to scheduling rather than an optimal OR method as follows:

> ...
> The first type of approach {the OR approach} was rejected on the basis that: {1} known future requirements and probable events will influence the short term schedule in an aggregate way, but it would be unreasonable to incorporate these into a short term optimization approach because of the uncertainty associated with them.
> {2} the environment is unlikely to remain static even over a period as short as 12 hours. (Kerr and Ebsary 1988, pp. 19-20)

A real life production environment is subject to changing constraints, fluctuating resource availabilities and even conflicts among the management objectives. As such, the problem is more of rescheduling than scheduling due to the dynamic nature of the problem (Rodammer and White 1988). OR models are not capable of capturing the important qualitative aspects of the problem as actually occurring on the shop-floor. Furthermore, most of the assumptions are never observed--almost no attribute of the system is deterministic--or are irrelevant to the real problem. Finally, real life scheduling problems have almost always no feasible solutions (Fox and Sadeh 1990).

---

[1]The original text is adapted from (Feigenbaum, McCorduck, and Nii 1988), pp. 56-57.

Since production must still continue, some constraints--we will call them soft constraints--must be relaxed in an intelligent manner so as to minimize both the cost of re-solving the problem and the possible cost associated with relaxing that particular constraint. For example, relaxing the due date constraint of a less important job or relaxing the constraint on labor availability by using overtime are common courses of action. One feasible way of achieving this is to have a dedicated knowledge base that can guide the relaxation process so as to minimize the associated costs.

2.1.4 Last Word on OR Approaches

Bel et al. summarize the above observations.

> Optimization by combinatorial methods. This approach is mainly concerned with a static point of view, and focuses on global objectives satisfaction. Nevertheless, due to algorithm complexity, it does not deal with realistic situations. As a result, the model and criteria represent only a very limited part of the real-live problems, which reduces the optimality and even the feasibility of the solutions generated. (Bel et al. 1989, p. 209)

There are successful OR approaches in some industries, such as the flight scheduling system of American Airlines (Parker 1989), but it hardly resembles the production scheduling problem as far as constraints and the importance of real time decisions are concerned. Even while dealing with the "nice" problems, there is another inevitable problem OR modelling approaches suffer, Baker bluntly declares:

> The traditional OR syndrome covering the application development cycle has five steps: 1) modeler sets up application, 2) model is demonstrated to work, 3) end user is trained, 4) modeler leaves, 5) application dies. This syndrome is especially prevalent with mathematical programming application. In fact, we might say that the use of mathematical programming has been limited by the availability of experts. Now, in

> many industries which have long relied on mathematical programming, these experts are taking early retirement. (Baker 1990, p. 108)

It is now clear that there is no easy way to solve all scheduling problems , since most of them are NP-complete. Those that can be solved in polynomial time can not fully nor closely represent the real life problem, and thus we can not draw conclusions from their results. What is the most promising approach then? If solving this problem is so hard, what is the remedy found in real manufacturing applications?

Researchers and manufacturers have found a few solutions for the computational burden associated with solving the scheduling problem. Instead of looking for a globally optimal solution, a good feasible solution that reflects the important management objectives can be accepted. There are several ways of finding such solutions: by simulation, by dispatching rule heuristics or by artificial intelligence approaches.

Optimization by simulation is preferred over optimization by analytical methods when the function can not be represented analytically, or an analytical solution is not possible. There are different approaches to optimization by simulation such as response surface methodology or perturbation analysis.

Response Surface Methodology (RSM) is used to define the relationship between the input variables and the performance criteria. Usually it is assumed that the performance criteria can be represented as a k^{th} degree polynomial function of the input variables. The technique requires repetitive simulation runs with varying input variable values. Once the response surface is estimated the optimal can be found by some search technique.

The Perturbation Analysis Methodology (PAM) uses an estimate of the gradient of the function of interest to direct its search. The input variables are perturbed slightly throughout the simulation run to get an idea of the gradient function. Depending on the new gradient estimate, more perturbation is done. Finally the simulation stops when the output of the simulation converges to an acceptable value. One advantage of this method over RSM is that optimization is done within the simulation, however, like all other gradient search techniques it suffers from convergence to local optima.

For scheduling problems, simulation is used to test the performance of certain dispatching heuristics, such as in Vepsalainen and Morton (1987). Usually the problem is fixed and a number of runs is performed to compare two or more heuristics in the same setting. There is no effort to discover under which parameter values the heuristic being tested is superior to the others.

In the following sections, we will discuss dispatching rule heuristics and artificial intelligence approaches.

2.2 Dispatching

A popular alternative to optimization methods is the use of dispatching rules to select the next job to be processed on a machine. This appears to be the actual practice used in many shop floors where the dispatcher uses a simple rule of thumb to select the next job. The dispatching problem can be defined as follows: given a processor and a queue of jobs waiting to be processed by that processor, find a dispatching rule or a set of rules that will satisfy the management objectives.

Dispatching rules have been found to be easy to use and thus very convenient for real time decision making. However, "a noticeable shortcoming is the inherent myopic nature of dispatching" (Bhaskaran and Pinedo 1992), pp. 1. The greedy nature of the rules causes suboptimal, low quality solutions for certain problems. They are useful for short term scheduling in an environment where exogenous events are highly likely to occur. Research on dispatching rules have shown that there is no dispatching rule that dominates the others on all types of problems or even on the variations of the same problem. For example, the SPT rule selects the job having the shortest processing time and solves the $1//\Sigma F$ problem optimally. However, for the same problem, the EDD rule selects the job having the earliest due date and is superior if the objective function is L_{max}. When the production environment is highly dynamic and when random events are common, dispatching rules perform better than they do in a deterministic environment. This observation is stated in Bhaskaran and Pinedo:

> ...
> In contrast, when the processing times of all jobs are deterministic and the machine is subject to an arbitrary breakdown process, the WSPT rule does not necessarily minimize the weighted sum of completion times. {On the contrary, in a stochastic, one machine environment where processing times are randomly distributed WSEPT rule minimizes the sum of the expected completion times.} Such a phenomenon, where simple dispatching rules are more robust in a random environment than in a fully predictable deterministic environment, is fairly common. (Bhaskaran and Pinedo 1992, pp. 11-12)

On top of being robust, dispatching rules have some other nice properties. It has been found--but not analytically proven--that the effectiveness of most dispatching rules is insensitive to the shop size and the arrival pattern. However, in minimizing lateness

and tardiness measures, related dispatching rules are somewhat sensitive to the due date assignment rules (Blackstone, Phillips and Hogg 1982).

In the following subsections we discuss several dispatching rules and present a large list of rules compiled from articles appearing in the literature.

2.2.1. Some Dispatching Rules Discussed in Literature

In this section some dispatching rules and the cases where they have been found useful will be listed. It also has to be noted that most of these rules are tested in a job-shop environment, but there is no inherent disadvantage in using them in a flowshop environment. We will not consider dispatching rules used for processes with precedence constraints since they do not have direct implementation for a flowshop environment.

Rules referred to as static do not change the priority of the job as time passes. On the contrary, dynamic rules change the priorities of jobs as time advances.

Even though some authors make a distinction between dispatching and sequencing rules, in this research they are considered the same. If the dispatching rule is applied N-1 times to a queue of size N, by removing the highest priority job each time, it acts as a sequencing rule.

Below we summarize many common dispatching rules.

The Select In Random Order (SIRO) rule is simplistic in that it does not use any information about the shop floor. As expected, it has not been found useful for any setting.

The First Come First Served (FCFS) rule assigns the highest priority to the first job in the queue. It has been found useful for minimizing the variance of the average waiting time if $p_j \sim f$, for all j. It also minimizes the expected average waiting time in the queue if p_j is independent and identically distributed (i.i.d).

The First At Shop First Served (FASFS) rule, a variation of FCFS used in job-shops gives the highest priority to the job that entered the shop the earliest, breaking ties arbitrarily or by FCFS. It has been useful for reducing the variance of the average total time spent in the shop by all jobs.

The Earliest Due Date (EDD) rule assigns the highest priority to the job with the earliest due date. The EDD rule minimizes maximum lateness, maximum tardiness and in the case of non-deterministic processing times, minimizes the expected maximum lateness but not the expected maximum tardiness if the environment is a single machine or a proportionate flowshop. Another version of EDD, referred to as Modified earliest Due Date (MDD) rule, tends to produce good tardiness results. The modified due date is computed as

$$d_j^* = \max(d_j, t + p_j)$$

where t is the current time (Baker and Bertrand 1982) . MDD has a tendency to perform like SPT when due dates are tight and like EDD when due dates are loose.

The Longest Processing Time (LPT) rule selects the job with the longest processing time. Even though it is not optimal it is useful in $mP//C_{max}$ problems. Another version of LPT, the Longest Remaining Processing Time (LRPT) rule, is optimal for $mP/pmtn/C_{max}$. If the processing times are stochastic, the Longest Expected Remaining

Processing Time (LERPT) rule is optimal for $P//E[C_{max}]$. LEPT and LERPT behave identical for a large family of problems.

The Most Work Remaining (MWKR) rule allocates the highest priority for the job with the most work remaining on it, where work is defined as $\sum_{i \in S} p_{i,j}$ and S is the index set of the remaining operations. MWKR tends to minimize the makespan for a job shop environment.

The Shortest Processing Time (SPT) rule is perhaps the most discussed rule in literature. SPT selects the job with the shortest processing time. SPT is optimal for $1//\sum C_j$, $mP//\sum C_j$, $mF//\sum C_j$ where machines are proportionate, and 1//J(t), mP//J(t) and mF//J(t), where J(t) is the number of jobs in the shop at time t. It is also optimal for 1//J and mP//J , mF//J where J is the average number of jobs in the system (Bhaskaran and Pinedo 1992). SPT also minimizes mean lateness in a single server environment. Even though SPT performs well in minimizing the number of tardy jobs when due dates are established exogenously (Blackstone, Phillips and Hogg 1982), one of the problems with SPT is that it causes high variance in lateness related measures (Day and Hottenstein 1970). It is also found to be irrelevant to cost-related measures, suggesting that cost is not related to flow-time (Blackstone, Phillips and Hogg 1982). In the case of stochastic processing times, and under fairly general conditions the Shortest Expected Processing Time (SEPT) is optimal for the above problems. The Shortest Remaining Processing Time (SRPT) rule minimizes the objective functions optimized by SPT for the single

machine problems, when jobs have different release times and preemption is allowed (Bhaskaran and Pinedo 1992.)

The Slack per Operation (SLACK/OPN) rule selects the job with the smallest slack time per operation ratio. The Minimum Slack (SLACK) rule selects the job with the minimum slack time remaining. SLACK/OPN, SLACK and EDD rules are known to perform better in minimizing the variance in lateness related measures compared to SPT and FCFS. SLACK/OPN performs better in minimizing the number of tardy jobs compared to EDD and the SLACK rules (Blackstone, Phillips and Hogg 1982).

One other due date related rule is the Critical Ratio (CR) rule, which selects the the job with the smallest critical ratio. The critical ratio CR is calculated by

$$CR = \frac{|d_j - t|}{L}$$

and L is the waiting time remaining in the queue for job j (Blackstone, Phillips and Hogg 1982). CR rule is known to work well for mean tardiness (Park, Raman and Shaw 1989).

The Least Work Remaining (LWKR) rule selects the job with the least work remaining, where work is defined as before. LWKR tends to minimize the flowtime in job-shops.

The Weighted Shortest Processing Time (WSPT) rule, assigns the highest priority to the job that has the highest $\frac{w_j}{p_j}$ ratio, where w_j is the weight--or importance--given to the job. WSPT is known to optimize the $1//\Sigma w_j C_j$ problem, and is known to work well

for parallel machines and proportionate flowshops. The WSEPT rule is optimal for $1//\Sigma w_j E[C_j]$ in a stochastic environment even in the presence of machine breakdowns.

The Shortest Setup Time First (SST) rule assigns the highest priority to the job that requires the shortest setup time, possibly zero time units if the previous job was from the same family. SST tries to minimize the total time spent in setups. SST has not proven very useful.

The following two rules try to capture the power of individual rules by taking a combination of them.

The Dynamic Composite Rule (DCR), attempts to minimize the total tardiness in a job-shop environment. DCR selects the job with the highest priority index $I_j(t)$ which is calculated as

$$I_j(t) = k_1 (d_{ij} - p_{ij}) + k_2 W_i(t) p_{ij} + \frac{k_3 W_h(t)}{\sum_{l=1}^{m} W_l(t)}$$

where m is the number of machines in the shop, and $W_i(t)$ and $W_h(t)$ are the current workload of the current machine and the next machine respectively. The performance of the rule is comparable to SPT in total lateness and the number of tardy jobs, and is better in total tardiness. It also produces low variance in flowtimes compared to SPT (Bhaskaran and Pinedo 1992).

Perhaps one of the most useful rules in a job-shop environment is the COVERT rule. The Cost OVER Time rule aims at minimizing the total tardiness. The priority index $I_j(t)$ is calculated by

$$I_j(t) = \frac{\max(k\bar{p} - u_j, 0)}{p_j k\bar{p}}$$

where $\bar{p}$ is the sum of the processing times of the jobs divided by the number of machines. The job with the lowest index is given the highest priority.

As can be seen from the above discussions no single dispatching rule dominates the others in all environments. On the contrary, most are very specific to a limited number of environments. Given the vast variety of scheduling environments and the changing nature of the job shop, it is not practical to find a feasible enumeration method to find out which dispatching methods to use under which circumstances.

Our research will try to overcome this problem by using a system responsive to changes. Our theoretical and applied method will adaptively update the set of current dispatching rules as the environment changes.

2.2.2 A List of Dispatching Rules

Before we present a list of dispatching rules we will introduce additional notation that is adapted from Blackstone, Phillips and Hogg, (1982). Let

- t be the current time;
- S_j be the remaining set of work stations before the job j is completed;
- I_{ij} be an indicator variable equal to 1 if job j is processed at station i;

d_{ij} be the time job j is scheduled to be completed at station i;

p_{ij}^{t} be the remaining processing time of job j on machine i at time t;

w_{ij}^{α} be the expected waiting time of job j in station (queue) i, given that there are α jobs waiting in line; and

DV_j be the dollar value of job j.

Definition 2.1: A proportionate flowshop is one where $p_{ij}=\alpha p_{kj}$, $\alpha>0, \forall j$.

Table 2-2 consists of a long list of dispatching rules. For each rule we give its name, abbreviation, the calculation of its priority index set, and conditions where it has been found useful.

Table 2-2 Dispatching Rules

NAME	ABBREVIATION	PRIORITY INDEX SET	COMMENTS
Select In Random Order	SIRO	U(1,N)	Uses no information about the shop.
First Come First Served	FCFS	1	Reduces the variance of average waiting time.
First At Shop First Served	FASFS	$\{j \mid r_j \le r_k, \forall k\}$	
Earliest Due Date	EDD	$\{j \mid d_j \le d_k, \forall k\}$	Minimizes L_{max}, T_{max} in a single machine or proportionate flowshop environment.
Longest Processing Time	LPT	$\{j \mid p_{ij} \ge p_{ik}, \forall k\}$	Good for minimizing makespan. For $mP//C_{max}$, the relative error is $\frac{4}{3} - \frac{1}{3m}$.
Longest Remaining Processing Time	LRPT	$\{j \mid p_{ij}^t \ge p_{ik}^t, \forall k\}$	Optimal for $P/pmtn/C_{max}$.
Most Work Remaining	MWKR	$\left\{j \mid \sum_{i \in S_j} p_{ij} \ge \sum_{i \in S_k} p_{ik}, \forall k\right\}$	Tends to minimize makespan in a job-shop.
Shortest Processing Time	SPT	$\{j \mid p_{ij} \le p_{ik}, \forall k\}$	Minimizes $\sum C_j$, the number of jobs in the system at any time, and average number of jobs for single machine, identical parallel machines and proportionate flowshop settings.

Table 2-2--continued.

NAME	ABBREVIATION	PRIORITY INDEX SET	COMMENTS
Shortest Remaining Processing Time	SRPT	$\{j \mid p_{ij}^{t} \le p_{ik}^{t}, \forall k\}$	Optimal for $1/r_j/f$, where f can be one of $\sum C_j$, the number of jobs in the system at any time, average number of jobs in a single machine.
Least Work Remaining	LWKR	$\left\{j \mid \sum_{i \in S_j} p_{ij} \le \sum_{i \in S_k} p_{ik}, \forall k\right\}$	Attempts to minimize the flowtime in a job-shop.
Weighted Shortest Processing Time	WSPT	$\left\{j \mid \frac{w_j}{p_{ij}} \ge \frac{w_k}{p_{ik}}, \forall k\right\}$	Optimal for $1//\sum w_j C_j$. Works well for parallel machines and proportionate flowshops for the same objective function.
Shortest Setup Time	SST	$\{j \mid s_{ilj} \le s_{ilk}, \forall k\}$	A greedy rule which tries to minimize the time spent in setup.
Weighted sum of SPT and LWKR		$\{j \mid \alpha p_{ij} + (1-\alpha)\sum_{i \in S_j} p_{ij} \le \alpha p_{ik} + (1-\alpha)\sum_{i \in S_k} p_{ik}$ $0 \le \alpha \le 1, \forall k\}$	
Remaining Processing Time / Imminent Processing Time	RPT/IPT	$\left\{j \mid \frac{\sum_{i \in S_j} p_{ij}}{p_{ij}} \le \frac{\sum_{i \in S_k} p_{ik}}{p_{ik}}, \forall k\right\}$	
SPT truncated to FCFS	SPT-FCFS	Use SPT first. If there is any item waiting more than some specified amount of time, select by using FCFS.	
FCSF truncated to SPT	FCFS-SPT	Use FCFS first. Whenever the queue length is above l_1 switch to SPT until the queue length reduces to l_0.	

Table 2-2--continued.

NAME	ABBREVIATION	PRIORITY INDEX SET	COMMENTS
	SPT^x	Calculate $F_j = d_j - t - \sum_{S_j} p_j - U$. Select $\{j \mid F_j \le F_k, \forall k\}$ if $F_j \le 0$ use SPT otherwise.	
	SPT-T	Use SPT + γ, where γ is a control parameter. Compare it to SLACK/OPN and choose the minimum.	
Minimum Slack Time	SLACK	$\{j \mid d_j - t - \sum_{i \in S_j} p_{ij} \le d_k - t - \sum_{i \in S_k} p_{ik}, \forall k\}$	
Slack / Operation	SLACK/OPN	$\left\{ j \mid \frac{d_j - t - \sum_{i \in S_j} p_{ij}}{\sum_{i \in S_j} p_{ij}} \le \frac{d_k - t - \sum_{i \in S_k} p_{ik}}{\sum_{i \in S_k} p_{ik}}, \forall k \right\}$	
Modified Earliest Due Date	MDD	$\{j \mid d_{ij} \le d_{ik}, \forall k\}$	Sometimes called the operation due date.
Modified Minimum Slack	MSLACK	$\left\{ j \mid d_j - t - \sum_{i \in S_j} p_{ij} - \sum_{i \in S_j} w_{ij}^{\alpha} \le d_k - t - \sum_{i \in S_k} p_{ik} - \sum_{i \in S_k} w_{ik}^{\alpha}, \forall k \right\}$	
Job Slack Ratio	JSR	$\left\{ j \mid \frac{d_j - t - \sum_{i \in S_j} p_{ij}}{d_j - t} \le \frac{d_k - t - \sum_{i \in S_k} p_{ik}}{d_k - t}, \forall k \right\}$	

Table 2-2--continued.

NAME	ABBREVIATION	PRIORITY INDEX SET	COMMENTS
Modified Job Slack Ratio	MJSR	$\left\{ j \middle\| \frac{d_j - t - \sum_{i \in S_j} p_{ij} - \sum_{i \in S_j} w_{ij}^{\alpha}}{d_j - t} \leq \frac{d_k - t - \sum_{i \in S_k} p_{ik} - \sum_{i \in S_k} w_{ik}^{\alpha}}{d_k - t}, \forall k \right\}$	
Greatest Dollar Value	GDV	$\{ j \mid DV_j \geq DV_k, \forall k \}$	
Critical Ratio	CR	$\{ j \mid \frac{\lvert d_j - t \rvert}{L} \leq \frac{\lvert d_k - t \rvert}{L}, \forall k \}$	Gives good results for mean tardiness.

2.3 Overview of Artificial Intelligence

Artificial Intelligence (AI) emerged as a symbolic computation paradigm during early 1960s. It was the aim of AI research to generate systems that could be considered intelligent. Even though intelligence in humans takes many forms, such as judgement, creativity, plausible reasoning etc., AI research restricts its attention to "computational techniques for performing tasks that apparently require intelligence when performed by humans" (Tanimoto 1990).

AI research mainly focuses on knowledge representation, developing search techniques, perception and inference (Tanimoto 1990). To date many applications have been developed and used that are products of AI research. Expert Systems, robotics, character recognition, natural language understanding, planning, learning applications are examples of the results of AI research.

Mathematical Programming methods in OR use optimization techniques to solve complex problems in real domains. Mathematical Programming methods have an objective function to be optimized and a constraint set that restricts the search space. On the contrary, AI methods depend heavily on heuristic knowledge to direct a search. The relationships in the problem do not need to be represented by mathematical equations. Constraints and objective function(s)--usually referred to as goals--can be ill defined and all may be non-quantifiable (Simon 1987). Most importantly, AI methods can incorporate experience and expertise into the solution of the problem.

Neither the Mathematical Programming approaches nor the dispatching approaches to scheduling problem have superior problem solving abilities as compared to each other.

AI approaches look promising for problems of higher complexity. A solution form an AI method may not be optimal and hence, may be inferior to the solutions found by optimization algorithms. However, the flexibility of modelling and the ability to yield reactive solutions is greatly enhanced. In dynamic production environments, the biggest problem a scheduler is usually faced with is how to handle exceptions such as rush jobs, machine breakdowns, resource unavailabilities, etc. Clearly mathematical models of the scheduling problems are seldom capable of handling such cases whereas most of the implemented AI systems, such as OPIS (Fox 1990), (Ow 1988), are able to handle these cases using heuristic knowledge.

In the following subsections, four different families of AI approaches to solving the scheduling problem will be presented. The distinction between any two families is not well defined since all methods borrow ideas from each other. Nonetheless each family captures the prevalent theme of its members. The four categories are:

1. Expert Systems (ES) approaches,
2. Constrained Heuristic Search (CHS),
3. Hybrid methods, and
4. Machine Learning (ML) approaches.

2.3.1. Expert Systems and Scheduling

Expert Systems (ES) attempt to mimic the decision making behavior of the domain experts. An ES requires three basic components:

1. a knowledge base,

2. an inference engine, and

3. an explanation subsystem, coupled with a user interface.

In the scheduling context, a knowledge-base has rules of thumb or facts that represent the knowledge about the scheduling problem, the factory environment, job descriptions, etc. Any inference strategy, such as backward chaining or forward chaining, can be used depending on the structure of the knowledge base. An explanation subsystem will give the line of reasoning behind the decisions made by the system.

Even though the ES approaches have been successful in many domains, like loan approval, medical diagnosis, fault detection, etc., the same has not always been true for the scheduling domain. A close look at the medical diagnosis domain, for example, will reveal the fact that there indeed exists domain experts from whom the necessary knowledge can be extracted. This however is not so clear for the scheduling domain. The next subsection will present views concerning this issue.

2.3.1.1 Cases against and for Expert Systems in Scheduling

Building a knowledge base for scheduling is a clear bottleneck for an expert scheduling system. In addition, there is considerable controversy about the existence of scheduling experts. However, Mc Kay et al. (1992) report that there are scheduling experts. They state:

> In conclusion, it is suggested that human expertise in scheduling exists and is a valuable component of any realistic scheduling planning environment. This expertise is like other expertise, and it is based on the human's learning to recognize, understand, and adjust for "broken-leg" cues or configural information found in the manufacturing environment. (Mc Kay et al. 1991, p. 24)

Still, most researchers have beliefs against the suitability of ES approaches to scheduling. Fox reports the following difficulties.

> ... taking an expert systems approach to scheduling appeared inappropriate. There are two problems with the expert systems approach:
> (1) Problems like factory scheduling tend to be so complex that they are beyond the cognitive capabilities of the human scheduler. Therefore, the schedules produced by the scheduler are poor; nobody wants to emulate their performance. {There indeed no expert scheduler exists.}
> (2) Even if the problem is of relatively low complexity, factory environments change often enough that any expertise built up over time becomes obsolete.
> Expert systems appear to be appropriate only when the problem is both small and stable. (Fox 1990, p. 81)

Basically expert systems are not suitable for the scheduling domain for the following reasons, (Fox 1990), (Rodammer and White 1988), (Blessing and Watford 1987), (Savell, Perez and Koh 1989) and (Steffen and Greene 1986).

1. Most scheduling experts are not real experts. Their decision making process is more akin to reacting to crises or management dictates rather than that of analyzing the underlying problem. Thus, they can not give general recipes for cost-effective solutions.
2. The scheduling problem is inherently very complex. Hence large-scale problems are hard to solve within a reasonable amount of time using inference strategies such as froward or backward chaining.
3. The scheduling environment is highly dynamic, so any rule of thumb valid today is prone to being obsolete tomorrow.

Contrary to the above results Miller, Lufg and Walker report a successful ES implementation.

> The large potential savings achievable through Expert Systems Scheduling, as proven by the $10 million annual pay back at a Westinghouse plant, will attract significant industrial interest in the next few year, making this one of the hottest areas for expert systems applications (Miller, Lufg and Walker 1988, p. 176-177)

Even though ESs are generally believed to be unsuitable for the scheduling environment, there is still a big advantage for expert systems ideas. Their ability to represent ill structured, ill defined, qualitative aspects of the problem is valuable. The inherent ability of ESs to represent such characteristics can be incorporated into a hybrid approach which will be discussed in the following sections.

2.3.1.2 Some Reported ES Applications

Most of the ES applications reported in this review are actually implemented or are in a prototyping stage. As expected, the authors report the standard bottleneck of building such systems--the knowledge acquisition stage. Even so, most of the applications are fairly successful with regard to acceptance by the staff and are being put to actual use on the shop-floor.

Clancy and Mohan (1990) describe an expert system, RBD, designed to help make real-time dispatching decisions by using sequencing rules representing company knowledge. To enable real-time decision making, the system is fed with real-time data about the factory floor status. Rules used for prioritization consider the following criteria:

. Critical ratio, start date, expiration time and lateness,

. Order type,

. Equipment status,

. Work station characteristics, and

. Job characteristics.

Using the above criteria the system produces a priority list of jobs. The highest priority job is then selected. The system has no learning capabilities.

RBD has been implemented and used in the semiconductor industry. The results obtained so far show a decrease in Work in Progress (WIP) and an increase in critical equipment utilization.

Kerr and Ebsary (1988) describe an expert system approach to the scheduling problem of a company producing "specialist products to customer specified configurations from approximately 3,000 different types of subassemblies, machine parts and raw materials."

Due to the complexity of the task, the authors were unable to elicit knowledge from the domain expert. When the classical knowledge acquisition techniques proved useless, the authors constructed the knowledge base by analyzing the actual schedules prepared by the schedulers. The resulting knowledge-base is composed of six different sets of rules categorized as follows:

. Priority rules,

. Job precedence rules,

. Forward loading rules,

. Dispatch rules,

. Contingency rules, and

. Time conversion rules.

The system was able to access real-time data about the shop-floor status via a relational database system, and make real time dispatching decisions.

The authors report two problems with the project.

. Capturing expert knowledge comprehensive enough to cover all the cases one may face in such an environment was impossible.

. The highly dynamic nature of the scheduling environment caused the knowledge base to be invalid very rapidly.

Lee and Suh (1988) describe an expert system, PAMS, with a hybrid knowledge-base composed of rules, frames and some heuristic knowledge which was specific to the domain of scheduling parallel machines. The knowledge base can be broken down into four sets of rules, dealing with:

. Job status,

. Machine status,

. Labor status, and

. Material status.

PAMS has a two stage control strategy where a feasible schedule is generated in the first phase and is evaluated and updated in the second phase, if desired. The system is capable of producing schedules at different levels of detail. The authors report improvement on tardiness measures.

Savell, Perez and Koh (1989) describe a classical Expert System application, implemented for a semiconductor production plant. Due to the modular nature of the environment the knowledge base could be constructed separately reflecting knowledge about each cell rather than the whole factory. The knowledge acquisition was carried out through interviews with the domain experts. The resulting knowledge base had two types of rules:

. Rules to assign priorities, and

. Rules to assign the jobs in production cells to the equipment (Equipment Scheduler).

Priority rules are used throughout the whole system but each cell has its own knowledge-base independent of the others for equipment assignment.

Even though the system helped to organize the current scheduling practice, the authors report the problems with run-time performance and the assessment of the system.

2.3.2. Constrained Heuristic Search

Constrained Heuristic Search, CHS, can be viewed as constraint satisfaction, where a search to find the feasible solution is guided by heuristic knowledge.

The constraint satisfaction problem, CSP, can be formulated by using a constraint graph, where nodes are variables and arcs are n-ary constraints among the values the variables may be assigned. Problem solving is performed by sequentially choosing a variable and a value to assign to it that satisfies all constraints incident upon it. As expected, the search for a solution is completed when a node is found where all the

constraints are satisfied and all variable assignments are made. If a node is found that can not be expanded yet no feasible solution is reached, the algorithm backtracks to the closest unexplored node and continues the search from there. In a scheduling problem this can be viewed as generating sub-schedules and backtracking when the current sub-schedule violates the constraints.

CHS utilizes the techniques used for CSP but, in addition uses heuristic knowledge to guide the search, so that costly backtracking is minimized. Perhaps the most crucial difference between CSP and CHS is that CHS also makes use of an objective function to evaluate the assignments, rather than a blind search that will result in poor solutions with respect to the performance criteria. Fox, Sadeh and Baykan (1989) present heuristic information that they found useful for the scheduling problem which they refer to as problem textures:

1. Value Goodness,
2. Constraint tightness,
3. Variable tightness with respect to a set of constraints,
4. Constraint reliance,
5. Variable tightness,
6. Variable contention, and
7. Constraint arity.

When viewed as an AI search algorithm, the CHS methodology consists of applying an operator that matches the current state and transforms it to a new state. Algorithmically the CHS can be described as follows (Fox, Sadeh and Baykan 1989):

1. Define an initial state, where no variable is assigned a value.

while the goal state is not reached do {

2. Match the best operator that applies to the given state.

3. Apply the operator.

}

The matching performed in Step (2) is the most difficult part of the algorithm. For a successful match the algorithm not only needs to satisfy the preconditions of the relevant operators, but also needs to find the best operator to apply in order to reduce the search complexity of the problem.

Note that Step (3) will result in either adding a structure to the graph, further restricting the domain of the variable, or reformulating the problem, such as relaxing some of the constraints.

In the rest of this section, we will present papers that use CHS as their main search algorithm. Even though most systems presented here use unique strategies to guide their search, the main search strategy is CHS. Most systems use additional heuristics other than CHS, to limit the search space and reduce the computational burden.

Arff and Hasle (1990) discuss constraint-guided heuristic search and the knowledge used to implement this methodology. A system called PLATO-PS is under implementation that uses the above ideas. The system has a process generation subsystem that generates routing and processing information. The sequencing subsystem "utilizes sequencing heuristics to transform the set of orders from a flat structure to a hierarchical structure of groups, possibly ordered". "The scheduler traverses the production order

sequence using heuristic, constraint-guided state space search". The sequencer utilizes heuristics for pruning the search space, such as:

. ordering heuristics and

. grouping heuristics.

The constraint-guided search uses heuristics for pruning the state-space search, such as:

. constraint violation and

. state selection.

Bensana et al. (1986), Erschler and Esquirol (1986), and Bel et al. (1989) introduce two systems called MASCOT and OPAL. The systems are designed to use three types of knowledge during the process of deriving feasible schedules:

. The theoretical knowledge on scheduling problems,

. Empirical knowledge about priority rules, and

. Practical knowledge about technological constraints.

The so called Constrained Based Analysis (CBA) approach aims at generating restrictions on local scheduling decisions by only considering limit times and resource availability constraints. As soon as a new precedence constraint is found, a limit time updating is searched. If such an update can be accomplished, the new value is propagated as far as possible, along the technological and other precedence constraints which have already been accounted for. When no more updating is possible, a search for a new precedence constraint is undertaken.

The OPAL system uses CBA to calculate the consequence of time constraints on the sequencing of operations, and generate new precedence constraints. A decision

support module is used to provide advice on the sequencing of operations based on practical or heuristic knowledge. Finally the supervisor coordinates the interaction between the two modules by calling each in order as needed. If no feasible schedule is found the supervisor calls a failure recovery module that selectively relaxes some of the constraints such as due dates.

Erschler and Roubellat (1989) introduce a system called OARABAID which uses constraint-based analysis as described above.

Constrained heuristic search has been extensively utilized for the implementation of the ISIS and CORTES systems. The general design of these systems are geared towards:

. constructing a representation language adequate enough to capture the nature of the problem and

. developing a search architecture capable of exploiting the search space both effectively and efficiently.

ISIS-1 is an order-centered scheduler which tries to construct schedules by considering the jobs rather than the available resources. The system identifies the next job to schedule through the use of CHS and tries to schedule each order and selectively relax any causal constraint found to be unsatisfactory (Fox 1984). ISIS-2 is an enhanced version of ISIS-1, where instead of solving the problem at one level, the system attacks it in a hierarchical manner. The systems first selects an order, then analyzes the available capacity, checks the resource constraints and finally assigns the time bound of each resource required for the operation (Fox and Smith 1984b). ISIS-3 introduces the idea

of resource versus order scheduling which is referred to as "multiperspective scheduling" by the author (Fox 1990). First, bottleneck machines are identified by a rough draft of the final schedule and these machines are tried to be assigned operations. Then the system considers an order based scheduling.

CORTES is a distributed system for production planning, scheduling and control (Fox and Sycara 1990). The CORTES system, compared to the ISIS family, considers jobs and resources as aggregate variables (Fox 1990). The approach used in activity scheduling is very similar to the least commitment strategy used in planning systems. The problem solving method of the CORTES system can be summarized as follows.

1. An initial state is generated.
2. The constraints imposed by the current state is propagated.
3. Texture measures (Fox, Sadeh and Baykan '89), are computed.
4. Based on (3), a state is selected.
5. A rule is matched to the current state, and an assignment is made. If the goal state is not reached, steps (2) through (4) are repeated.

Research on both ISIS and CORTES has reported promising results. The use of texture measures has resulted in a reduction of the search complexity.

Steffen and Greene (1987) introduce an approach named "dedicated-shared decomposition," where, after the production planning level, they assign certain jobs to dedicated machines, and then in each subproblem they perform dedicated loading, sequencing and shared loading and sequencing.

Usually selection of the dedicated machine is done according to the customer preference. The scheduling is handled by a heuristic algorithm that will minimize the conflicts for competing jobs. Loading and sequencing of the shared resources are done by "Constraint Directed Search." Based on adherence to the constraints included in the prototype, the authors conclude that the approach is applicable.

2.3.3. Hybrid Artificial Intelligence Methods

The OPIS family is different from the ISIS/CORTES family in that it uses a blackboard style control. The blackboard contains the necessary information about the factory and is capable of managing the temporal constraints.

The actual scheduling decision are made by an analysis and two decision knowledge sources (KSs). The former one is used to analyze the available capacity whereas the latter two are used for resource or order based scheduling decisions.

At each cycle, after propagating the results of any unexpected events such as capacity conflicts, machine breakdowns, etc., the search manager prioritizes and stores them on the "agenda." The task with the highest priority is executed as a subtask (Fox 1990).

OPIS 2 is a reactive scheduling system which is designed to deal with exogenous events. When such an event is encountered, the consequences of the effects are propagated onto the existing schedules. Conflict arising from this change are stored in the "agenda" as is the case in OPIS 1. Two more KSs are used to deal with the exceptions: the order based right shifter and the demand swapper. The system uses a

decision tree to decide which KS(s) to use for exception handling. The OPIS systems family extends the multi-perspective scheduling view of the ISIS 3 system and allows alternative solution methods within each view.

Ow, Smith and Howie (1988) describe a distributed scheduling system called CSS which uses two types of knowledge bases:

. the work order manager, WOM,

and

. the resource broker.

The idea of CSS is similar to the one described in (Shaw 1986). The system uses a bidding methodology. WOM estimates the completion time of a work order by examining its operations graph and initiates a bid among the nodes which could perform the job. After evaluating the bids submitted, the job is sent to the selected node. The resource brokers evaluate the availability of their resources and bid according to this information. The search method used in the system is called a two-pass best-first search.

Shaw (1986) uses a network-wide bidding scheme, where each cell bids for a job to schedule, and the one that has the best bid gets the job to process.

Decisions to be made by the system are:

. which cell to assign the job and

. the sequencing and scheduling within the each cell.

Performance measures used are:

. job flowtime,

. proportion of the jobs failing to meet the due date,

. job lateness and tardiness statistics, and

. average in-process waiting time.

The author uses a planning algorithm for scheduling within the cell.

Shaw (1988b) describes a decentralized, cooperative system where the manufacturing environment has been divided into cells composed of a number of machines. The cells can communicate with each other through a Local Area Network. The scheduling problem is addressed in two levels. In level one a cell that wants to submit a job calls for "bids" from the other cells which believe that they can process the job. Whichever gives the best bid with respect to the performance criteria wins the bid and the job is transferred to that cell.

At the cell level, a planning algorithm is used which is driven by pattern matching and state transformation operators. Basically the operators that match the current state are applied until a goal state is reached or a failure is reported--i.e. there is no infeasible solution. Cell levels also contain manufacturing modules (operators), decision rules, and scheduling heuristics.

The author reports that the system, on the average, performs better than myopic SPT, bidding SPT, and bidding EFT.

Shaw (1988c) describes a system that employs:

. pattern-directed inference to capture the dynamic nature of the scheduling environment,

. a planning technique to coordinate manufacturing processes and resource assignments, and

. the A* search algorithm (Tanimoto 1990), to expedite the searching procedure.

The author employs a strategy that generates a schedule for each job and then uses the planning methodology to resolve the conflicts that arises due to sharing the same resources at the same time intervals. To solve the conflicts the author suggests two approaches:

i. Critic Mechanism: This method uses a table called "Table of Multiple Effects (TOME)" where for each element in that table they have to check the delator lists and adder lists--which is very costly. Another drawback of this problem is that it can not identify alternative resources and operators.

ii. Reasoning about Resources: This works by identifying conflicting interactions and then, using a plan revision procedure tries to improve the makespan as much as possible.

In all three papers by Shaw, the author analyzes the feasibility of the bidding scheme used in connection with a planning algorithm at the cell level. The real schedule construction done at the cell level is a good example of heuristic search where cumulative processing time plus estimated total remaining processing time is used as heuristic knowledge to guide the search.

2.3.4 Machine Learning Approaches

None of the system described in the previous sections had a capability of discovering rules or strategies. Also none had mechanisms to improve their performance

as new decisions were made. The intelligence (the heuristics) was coded into the source as executable instructions. Unless the code changes, they will remain the same and will react exactly the same way they had done previously even if those actions resulted in a bad decision.

In this section we will present papers that use at least one Machine Learning (ML) approach to construct their knowledge bases. We will classify them as knowledge discovery systems. Note however that, once the knowledge bases are constructed, they are not updated. So even these systems can not improve their performances through "experience" with the possible exception of system SCHEDULE by Lamatsch et al. (1988). Their contribution is however their ability to construct knowledge that often helps the performance system to achieve better results than those accomplished by non-learning systems.

Blessing and Watford (1987) describe a FMS scheduling system, INFMSS, where the knowledge is represented by frames. They structure the knowledge base in three dimensions namely;

- FMS description on one axis,
- part mix on another,

and

- schedule proposed on the third axis.

If any point in that three dimensional space is already defined, the expert system uses that point as the best schedule. Otherwise a search for a few scheduling strategies for the given part mix and FMS description is carried out, and evaluated by a simulation system.

The schedule that gives the best result is chosen as the answer. Those points are saved for further reference.

This mechanism is referred to as rote learning. Even though the system saves old decisions that gave good results, it has no means of generalizing them and we suspect that in such a complex environment the chances of observing the same state is very low.

An interesting approach to the scheduling problem has been discussed by Chryssolouris, Lee and Domroese (1990). The objective of the study is to be able to find the weights of the criteria for the decision-making process at the workcenter level based on some performance measure. A Neural Net has been used for this purpose. Examples for training are generated by simulation by fixing the operational policy and workload of the workcenter. The outputs of the simulation are the performance measures. The input to the neural network is the workload of the workcenter and the desired levels of performance. The outputs of the network are the weights necessary to achieve the desired level of performances. The weights can be considered as follows: Given a scheduling heuristic, what must be the relative importance of local criteria with respect to each other so as to achieve the desired level of performance. The authors use a system called MADEMA to carry out the actual scheduling decisions within the simulation.

Hilliard et al. (1987), (1989) describe a series of experiments to learn general rules for simple job shop scheduling tasks by using classifier systems. The system knows different predicates, dispatching rules, which it can use for sorting the jobs in the queue. They expect the classifier system to find the best dispatching rule for ranking the jobs in the queue, with respect to the performance criteria. The authors report that the system

can discover the optimal rule, SPT, for a single machine minimum lateness problem (we assume this is total lateness), and good solutions for minimum weighted tardiness problem i.e. $1//\sum w_i T_i$.

Another emerging approach to scheduling problems is case based reasoning. The learning system stores each case seen so far as a piece of knowledge, in what is referred to as a "Case Base." During problem solving, if a match is found for any of the cases stored in the case base, the solution method or the results stored with the case is applied to the problem. The match does not need to be exact. Koton (1989), describe a system called SMARTplan that is used for resource allocation and scheduling for airlift operations. Due to the size of the search space only a partial match of the cases are required for successful instantiation of a stored case. Only those cases that are successful are stored for future use.

Lamatsch et al. (1988) describe an expert system named SCHEDULE that uses reduction digraphs to match the given problem as an instance of a scheduling problem solvable by the stored heuristics. All the heuristics are polynomial time algorithms that can produce optimal or near optimal results published in various journals. If no match is found, the system first tries to find a problem that is close to the original problem with respect to a metric they define. If that fails, a relaxation of the problem is solved. The system has a learning capability by adaptively adjusting the distances between two nodes of the graph. As the user prefers a problem to solve as a substitute for the original problem, the arc weight is reduced. By this way the system learns which scheduling

problem is a better approximation for a certain type of problem. Note that each node is a type of problem that can be reduced to each other if there is a path between them.

Nakasuka and Yoshida (1992) describe another inductive learning method to discover rules for a scheduling environment. At any given decision point the system is simulated forward by using all possible dispatching rules separately. Once the best rule is found, that state and the rule applied is saved as an example and the simulation continues until the next dispatching decision point. After enough examples are generated a binary decision tree is constructed by a specialized induction algorithms. The authors also describe the details of the induction algorithm. The system is used to find good solutions for the tardiness and makespan performances, i.e. one measure is given as a constraint and the other is optimized. The system performs better than single dispatching rules.

Piramuthu et al. (1991) describe a hybrid method for scheduling that employs an experimental environment, using simulation, and a learning component. New knowledge is incorporated into a pattern-directed scheduling system. For a given state defined by eight control parameters, the system is simulated by using each dispatching rule they have considered as useful. The dispatching rule that yields the best result with respect to the performance measure is specified as a positive example for the learning subsystem. In this way a training set is constructed. From the training examples a decision tree is constructed using Quinlan's ID3 algorithm. Each path from the root to the leaf of the tree gives a conjunction of the parameters, thus yielding a rule in conjunctive form. The leaf nodes describe the action to be taken. Once the learning part

is done, the system is ready to make schedules by matching the state of the system to one of the rules created and by using the dispatching heuristic at the leaf node to choose the operation to be performed.

Shaw, Raman and Park (1991) discuss adding a learning capability to a knowledge-based scheduling system. The authors discuss how rule induction can be used in conjunction with dispatching rules (i.e., rule induction can be used to determine the important attributes of the system and then to choose the dispatching rule to use). This is called Pattern Directed Scheduling (PDS) by the authors. In an experiment, the authors use four such rules that are derived by taking into consideration eight attributes of the system.

Even though the overall performance of PDS is reported to be better than using a single dispatching rule, this approach is very costly since examples have to be created by simulation runs except for a few known cases when an optimal rule is known to exist. Except for a few trivial scheduling problems, it is not possible nor feasible to construct or simulate an "oracle" that is required by most inductive learning algorithms.

Thesen and Lei (1986) describe a knowledge-based robot scheduling system. Knowledge is not acquired from a domain expert, but rather created using simulation experiments. The authors simulate the system by using dispatching rules and then, based on the specified performance measures, they decide on which dispatching rule to use given the system state which is defined by the robot position, number of electroplating tanks, etc.

Yih (1990) describes an approach to the knowledge acquisition task to schedule the operations of a material handling robot. The robot has to carry each part from one processing tank to another. This has to be scheduled so that the part do not get spoiled in the tanks. The authors develop the Trace Driven Knowledge Acquisition (TDKA) methodology to acquire rules. The methodology requires simulation experiments with scheduling experts, in this case students. The records collected from the simulation, traces, are used to define the classes and to select a rule applicable for each class. The author describes an algorithm for the class formation and rule assignment. Even though TDKA is suitable for knowledge extraction the method is limited by the existence of expert schedulers and their consistency during decision making. The method also suffers from heavy involvement with manual procedures.

Yih and Thesen (1991) describe the same experimental setting discussed in (Yih 1990). This time using the traces of the decisions a set of states and state transition probabilities are defined. Based on this information the decision process is analyzed as a Markov decision process behaving according to the calculated transition probabilities. The optimal policy of the model is used for rule formation. The idea is applied to the same problem described in Yih (1990). The method produces a good set of optimal rules, but they are not sufficient to cover the entire rule space and also they are limited with the existence of real scheduling experts. This shortcoming has been fixed in Yih (1992). After the optimal policies are discovered, the author uses a modified version of the algorithm described in Yih (1990), to generalize the optimal policies to cover whole rule

base. The rules formed this way perform much better than the identified human schedulers.

Yih, Liang and Moskowitz (1992) use the same problem to describe an OR/NN hybrid approach. This time an NN is used to generalize from the optimal policy of the semi-Markov process. That is, each state,optimal policy pair is given as an example to a NN. After training the NN is able to make predictions that are of better quality than human schedulers.

2.3.5 A Listing of Systems

In Table 2-3 we summarize the research done in knowledge based scheduling. Below we describe the ideas behind the columns in the table.

Knowledge Representation refers to the type of knowledge representation scheme used in the system.

For a feasible running performance most systems employ an intelligent search heuristic. Given the complexity of the scheduling domain this appears to be the key aspect of the system. As noted earlier, most ES applications presented here have run-time performance problems, due to using backward or forward chaining, without making use of other available information. However, using heuristics like CHS, often improve their performance.

The manufacturing environment for which the system is developed will be listed under the "Application" column.

Since almost all of the systems presented here have knowledge bases, it is also important to discriminate how their knowledge base is organized. This often reflects the problem solving philosophy of the system. For example, ISIS-2 uses a hierarchical problem solving architecture hoping that this will reduce the size of the search space. Some others facilitate a distributed problem solving architecture. The "Structure" column captures this information.

The ability of the system to learn is mentioned in the "Learning" column. If the system has a learning capability the nature of the learning algorithm used is listed.

Finally, the "External Module" column tells if the system is interacting with any other external systems. The interaction may be in the form of feeding information to an external module, communicating with a simulation program, getting data from a data base, etc.

Some columns of Table 2-3 are adapted from Kusiak and Chen (1988).

Table 2-3 Knowledge Based Systems

Author (System)	Knowledge Representation	Control Strategy	Application	Structure	Learning	External Module
Adachi, Talavage and Moodie (1989), (RBDSS)	Frames and Rules	Heuristics	Production Control	Tree structured; Multi-criteria objective	Statistical Inference	Simulation
Alpar and Srikanth (1989a), (ESCH)	Predicates	Backward chaining	Scheduling for blending machines (Cereal)	Four set of rules		
Alpar, and Srinkanth (1989b), (ESCH)	Predicates	Backward chaining, depth first search	Scheduling for blending machines (Cereal)	Four set of rules	Interactive Knowledge Acquisition	LP, Spreadsheet
Arff and Hasle (1990), (PLATO-PS)	Rules and heuristics	Constraint guided search	Scheduling			
Bel et al. (1989), (OPAL)	Object based, rules and semantic nets	Constraint based analysis, best first search with backtracking	Job Shop Scheduling			OR Modules
Ben-Arieh (1986)	Predicates and rules		Shop floor scheduling	Two level Structure		Algorithms for calculating costs etc.
Bensana et al. (1986)	Rules	Meta-rules and constraint based analysis	Job-shop scheduling			Rule based decision support module
Blessing and Watford (1987), (INFMSS)	Frames		Scheduling		Rote learning	Simulation for evaluation

Table 2-3--continued

Author (System)	Knowledge Representation	Control Strategy	Application	Structure	Learning	External Module
Brown (1988), (CAMPS)	Rules and frames	Constraint guided	Scheduling			DBMS
Chang (1985)	Rules	Forward Search	Job Shop Scheduling			Simulation
Chyrssolouris, et al. (1988), (1990), (MADEMA-S-DATA Component)	Rules		Production Scheduling	Hierarchical		
Chyrssolouris, Lee and Domroese (1990)	Neural Nets		Workcenter Scheduling		Learning by Examples	Simulation, MADEMA
Clancy and Mohan (1990), (RBD)	Rules		Job and Flow Shop Dispatching			Consistency checking module
Conway and Maxwell (1986), (LLISS)		Partly by human scheduler, menu driven	Interactive scheduling			
Carnegie Group (CORTES)	Frames, rules and heuristics	Micro-opportunistic constraint guided search	Scheduling	Distributed and Hierarchical		Simulation
Czigler and Whitaker (1990), (SHOPS)	Predicates	Heuristics and inference engine				
De (1988)	Predicates	Planning with backward deduction	Scheduling in a FMS	Two phased		
De (1990)	Frames	CPA and filtered beam search	Scheduling in a FMS	Three sets of knowledge		

Table 2-3--continued

Author (System)	Knowledge Representation	Control Strategy	Application	Structure	Learning	External Module
Erschler and Esquirol (1986), (MASCOT)	Rules	Meta-rules and constraint based analysis	Job-shop scheduling			
Erschler and Roubellat (1989), (ORABAID)	Rules	Constraint-based	Job Shop Scheduling			
Farhoodi (1990), (OCS)	Frames and rules	Algorithms, heuristics	Scheduling	Modular		
Fox (1983)	Frames	Constraint directed search	Large scale job shop scheduling	Three level hierarchy		
Fox and Kempf (1987)		Constraint based analysis	Scheduling	Two phased		
Grant (1986)	Frames and rules	Heuristic rules	Repair job scheduling	Three level hierarchy		
Carnegie Group (ISIS-1)		Constraint guided search	Scheduling			
Carnegie Group (ISIS-2)		Constraint guided search	Scheduling	Hierarchical		
Carnegie Group (ISIS-3)		Constraint guided search	Scheduling	Hierarchical, multiperspectiv e		
Jain and Osterfeld (1989), (ESS)	Frames	Backward simulation	Scheduling			Expert System embedded in Simulation

Table 2-3--continued

Author (System)	Knowledge Representation	Control Strategy	Application	Structure	Learning	External Module
Kak, et al. (1986)	Frames		Sensory based Robotic Assembly			
Kerr and Ebsary (1988)	Rules	Forward chaining	Scheduling assembly network	Six set of rules		Embedded in a relational DBMS
Lamatsch et al. (1988), (SCHEDULE)	Rules, algorithms	Problem matching and reduction	Job-shop scheduling		Arc weights of the reduction digraph is updated (discussion)	OR Scheduling Algorithms
Lawrence and Morton (1986), (PATRIARCH)	Heuristics	Heuristic prioritization		Hierarchical and distributed		
Lecocq and Guiot (1988), (CROPS)	Rules embedded in HPN	Plan generation	Scheduling in a FMS	Three level hierarchy		Simulation for validation
Lee and Suh (1988), (PAMS)	Rules, frames and heuristics	Forward chaining	Parallel machine scheduling	Hierarchical		DBMS
Lee (1987)	Predicates		Production Scheduling			
Martinez et al. (1988)		Petri-net and constrained heuristic search	Real-time scheduling and coordination	Four level of hierarchy		Simulation

Table 2-3--continued

Author (System)	Knowledge Representation	Control Strategy	Application	Structure	Learning	External Module
May et al. (1991), (MULTEX)		Backward and forward chaining, Mediator	Scheduling			OR Components Heuristic pricing subsystem
Mohan, Clancy (1989), (SIS)	Rules	Heuristics	Dispatching on the shop floor			Real-time on-line data base COMETS
Muller, Samblackx and Matthys (1987)	Rules		Scheduling in electroplating industry			Extracting knowledge by use of simulation
Newman and Kempf (1985)	Rules		Robot Scheduling	Rule base is divided into six levels		
Niew, Lim and Ho (1990)	Heuristic rules and frames	Constraint driven goal satisfaction	Master scheduling for IC's			
Ogrady and Lee (1988), (PLATO-Z)	Rules, flavours, procedures and dispatching rules	Multi-blackboard/actor model	Production cell control	Decentralized		
O'Keefe (1986)	Rules	Backward chaining	Simulation Construction			
Carnegie Group (OPIS-1)		Opportunistic, constraint guided search	Scheduling	Hierarchical, blackboard architecture		

Table 2-3--continued

Author (System)	Knowledge Representation	Control Strategy	Application	Structure	Learning	External Module
Carnegie Group (OPIS-2)		Opportunistic, constraint guided search	Reactive scheduling			
Ow, Smith and Thriez (1988)		Constraint directed search and conflict analysis	Scheduling			Simulation for performance evaluation
Ow , Smith and Howie (1988), (CSS)		Two pass, best first search with bidding	Scheduling	Distributed, cooperative		
Parunak (1987)		Constraint directed search	Distributed manufacturing system	Distributed AI system		
Piramuthu et al. (1991)	Rules	Pattern directed search	FMS and flow system scheduling		Inductive by ID3	Simulation
Powner and Walburn (1990)	Rules and frames		Scheduling in a FMS (Aerospace)		Automated knowledge acquisition (Suggested)	FMS Simulation
Raghavan (1988)	Rules represented as decision tables		Due date scheduling in a FMS		Discovering rules by observation, not automatic	Simulation
Ready, Simmonds and Taunton (1990), (PROSPEX)	Rules		Job-shop scheduling support tool			Simulation

Table 2-3--continued

Author (System)	Knowledge Representation	Control Strategy	Application	Structure	Learning	External Module
Reinschmidt, Slate and Finn (1990)	Rules	LP and inference engine	Scheduling	Four sets of rules		LP
Sarin and Salgame (1989)	Frames and rules	Blackboard, meta level knowledge	Scheduling	Independent knowledge sources		
Savell, Perez and Koh (1989)	Rules		Scheduling in a semi conductor plant	Distributed knowledge base for each cell		
Shaw (1986)	Predicates	Bidding and planning	General job-shop scheduling	Distributed and hierarchical		Simulation
Shaw (1988b)	Predicates	Bidding and planning	Scheduling in CIM environment	Distributed and hierarchical		Simulation for performance evaluation
Shaw and Whinston (1986)	Predicates, frames and rules	Heuristics and rules	General job-shop scheduling problem	Hierarchical problem solving		
Shaw (1988c)	Transformation operations	Pattern Directed	FMS Scheduling			
Shen and Chang (1986)	Frames and rules		General job-shop scheduling			Algorithms for detail scheduling
Shen and Chang (1988)	Frames and rules		FMS Scheduling			Scheduling Algorithms
Steffen and Greene (1986a)	Predicates	Constraint directed search with backtracking	Parallel processors in a FMS	Five level hierarchy		

Table 2-3--continued

Author (System)	Knowledge Representation	Control Strategy	Application	Structure	Learning	External Module
Steffen and Greene (1986b)	Predicates	Hierarchical planning and constraint directed search	Scheduling	Hierarchical		
Steffen and Greene (1987)		Heuristics and constraint directed search	Parallel machine scheduling	Hierarchical		GUESS/1
Subramanyam and Askin (1986)	Rules		Real-time job shop scheduling	Three level hierarchy		
Sullivan and Fordyce (1990), (LMS-DDM)	Function-object (FORK)		Semiconductor Fabrication Scheduling			
Thesen and Lei (1986)	Rules	Meta-rules	Robot scheduling Electroplating Line		Inductive by simulation	Simulation for performance evaluation
Warwick and Walters (1990), (MOPPS)	Rules	Meta-rules	Work cell scheduling	Hierarchical knowledge base		
Worrall and MacDonald (1987)	Dispatching rules		Scheduling			
Woodhead, Dobolyi and Pennington (1986)	Rules, Semantic Nets, Relational DBMS	Query Language	Process Planning			
Yang and Jiang (1990)	Rules	Backward, forward chaining combination	Monthly production planning (Chemical Plant)			DBMS

Table 2-3--continued

Author (System)	Knowledge Representation	Control Strategy	Application	Structure	Learning	External Module
Yih (1990), (TDKA)	Rules		KMS scheduling - Circuit Board Production		Learning by simulation	
Yih (1992)	Rules		FMS scheduling		Learning from examples	Simulation
Yih, Liang and Moskowitz (1992)	Neural Network		Scheduling		Learning from examples	Simulation

CHAPTER 3
LEARNING

3.1 An Overview of Machine Learning

Learning is a very broad term that denotes the way in which people increase their knowledge and skills. Machine Learning (ML) concentrates on generating systems that can learn and thus acquire new "declarative knowledge", improve their performances for problem solving through instruction or practice, and organize and generalize new knowledge (Carbonell, Michalski and Mitchell 1983). For example, the knowledge base required by an expert system can be generated by using learning algorithms and then refined by the learning sub-system so that the expert system performs better than the "no-learn" case.

Research on ML dates back to the early 1960's (Cohen and Feigenbaum 1982). Most of the ideas used today were initiated during the earlier years of ML research. For example, adaptive systems depending on random mutation and selection were created hoping that these processes would result in intelligent systems. Various analogs of neurons were developed and tested.

During the mid 1960's and early 1970's, it became apparent that learning was a hard task and that it was virtually impossible to learn without any prior knowledge of the domain or meta knowledge about the task. This led to research concentrated on systems that utilizes symbolic concept-acquisition (Carbonell, Michalski and Mitchell 1983).

Finally, within the last decade, ML research centered around the theory of learning as well as its applications to specific domains. Even though the theoretical approaches consider the most general form of the learning problem, most of the applied ML research focuses on problems specific to the domain for which it is being developed. Carbonell, Michalski and Mitchell describe the trend that started mid 1970's as " The Modern Knowledge-Intensive Paradigm."

Before discussing the details of ML research, it is useful to consider the dimensions of ML approaches. That is, we will characterize the approaches with respect to their assumptions. We use four dimensions of learning: definition, methodology, nature and mode. We will discuss the research done in ML within this four dimensional framework.

The first dimension, the definition of learning, describes how ML research perceives learning. What do researchers mean when they use the word "learning"? Not all applications, share the same view of learning. Let us see how learning is defined.

Cohen and Feigenbaum define learning from four different perspectives.

> {1} Herbert Simon (in press) defines learning as any process by which a system improves its performance... {2} A more constrained view of learning, adopted by many people who work on expert systems, is that learning is acquisition of explicit knowledge... {3} A third view is that learning is skill acquisition... {4} a forth view of learning is that it is theory formation, hypothesis formation and inductive inference. (Cohen and Feigenbaum 1982, pp. 326-327)

Most AI applications, regardless of whether they are learning applications or not, implicitly consider {1}. The distinction arises based on whether the system does it itself or it is hard-coded.

The acquisition of explicit knowledge is the main focus of ES applications. Even though ESs are not learning systems themselves, the problem of knowledge acquisition initiated a substantial amount of research on issues about learning. It is useful to consider the evolution of approaches to knowledge acquisition since we believe it demonstrates in which direction ML research is going.

It is not an exaggeration to say that ESs enjoyed success in domains where expertise existed and where knowledge did not become obsolete in time. This is exactly one of the reasons why ESs were not successful in the scheduling domain. The scheduling domain is so complex and dynamic that it is almost impossible for humans to understand the existing causal relationships. However, eliciting expert knowledge is the bottleneck of ES development (Cohen and Feigenbaum 1982), indeed in any knowledge intensive systems. After realizing this, the trend in knowledge acquisition shifted from classical techniques like protocol analysis, or interviews to interactive techniques where the knowledge acquisition task was given to the learning system. The learning system guided its search for concept formation by asking questions to the expert and then refining its current hypothesis based on the answer given by the expert. One such system is TEIRESIAS by Davis (1982) which is used for medical diagnosis.

During the course of development of ESs, researchers realized, to their dismay, that for some domains even interactive systems proved infeasible. There were no experts or the experts were not able to dedicate time to the knowledge acquisition process. This led to research in learning systems where the system itself can learn without the expert's

help. As a result, ML research concentrated on developing learning algorithms that are in turn used for developing systems that can learn.

Learning requires an extensive amount of inference. Inference can be deductive, or inductive, and sometimes both. During the inference process, the learning system needs to explore the hypothesis space, which is usually very large. This necessitates a powerful inference mechanism. There are different approaches to implementing this mechanism which are summarized within the second dimension.

The second dimension of learning is the philosophy employed for learning. In other words, it is the methodology utilized by the system to learn.

Most learning methodologies considered in ML literature can be classified into four groups: rote learning, learning by being told, learning from examples and learning by analogy (Cohen and Feigenbaum 1982).

Rote learning can be viewed as memorization of problem instances. The learning system does not try to generalize. An instance encountered is stored as knowledge. The main issue here is organizing the instances in storage and accessing them as needed. For example, assume the system is trying to learn the prices of different cars. Assume the system conceptualizes a car by using attributes such as color, A/C, number of doors, engine size, transmission type, etc. When the system is presented with the specifications of the car it checks to see if it has seen that instance before. If not, the system calculates the price of the car and stores the example. If so, the system responds without executing the calculation steps. One of the biggest disadvantages of rote learning is that the system has no means of identifying the fact that color is an irrelevant factor in price calculation

(assuming that is the case) and tries to calculate the price if presented a yellow colored car even if it has information about a red colored car of the same type.

Cohen and Feigenbaum explain learning by being told, or advice taking, as follows: "Learning by being told, in which the information provided by the environment is too abstract or general and, thus the learning element must hypothesize the missing details" (Cohen and Feigenbaum 1982), pp. 328. The system should have the knowledge to be able to interpret high level information, make it operational for itself, and then integrate it to the knowledge base (Cohen and Feigenbaum 1982). In a scheduling environment where the system is learning how schedules are evaluated, the system should have the capability of understanding what "avoid creating high WIP inventories" means and take the correct action to achieve this goal. This advice is then integrated into the existing knowledge base and the system decides if this is a useful advice after evaluating its performance. Note that such a system has very demanding assumptions like prior knowledge of the domain, consistency of the initial knowledge base, and an existence of a performance evaluation entity.

Cohen and Feigenbaum (1982), describe learning by analogy as follows:

> Learning by analogy, in which the information provided by the environment is relevant only to an analogous performance task and, thus, the learning system must discover the analogy and hypothesize analogous rules for its present performance task. (Cohen and Feigenbaum 1982, p. 328)

AM is a learning system that discovers concepts in mathematic and set theory. During its search, AM uses "reasoning by views or by analogy", which is one of its thirty heuristic rules (Cohen and Feigenbaum 1982). The ANALOGY slot of the concept frame

gives vague information about how a particular instance is an instance of another concept. AM tries to utilize this information together with the information supplied in the VIEWS slot to map existing examples into examples of the concept under construction (Cohen and Feigenbaum 1982).

Perhaps the most extensively analyzed learning strategy is learning from examples. There are very different approaches to learning from examples, such as learning by induction, explanation based generalization, Neural Networks (NNs) and Genetic Algorithm (GA) based learning systems. We will focus on induction and learning from examples in this section and only briefly mention neural networks. The GA based systems will be covered in Chapter 4.

Inductive learning is a data intensive approach (Quinlan 1986), where the learning task is to generate a knowledge base after seeing a set of examples. An example is an n-ary tuple representing n-1 attributes of the object of interest and its class. The resulting knowledge base should classify all of the training examples correctly. Most of the learning systems such as CLS, ID3, ACLS, ASSISTANT use decision trees for knowledge representation and others like AQR, CN2 try to learn production rules (Clark and Niblett 1989). A more general knowledge representation is decision lists (Rivest 1987).

Omitting the technical details, a decision tree construction algorithm can be stated as follows:

1. If all the instances are from exactly one class, then the current node is the answer node classifying all the examples.

2. Otherwise

a. select the attribute that is best for classifying the current examples.

b. us the attribute to partition the current examples adding a branch to the tree for each partition. Construct the decision tree recursively from each of these branches.

For example, assume a customer is represented by four attributes: age, salary, occupation and credit history. The learning task is to differentiate the good and bad customers for loan purposes. A decision tree may look like in Figure 3-1:

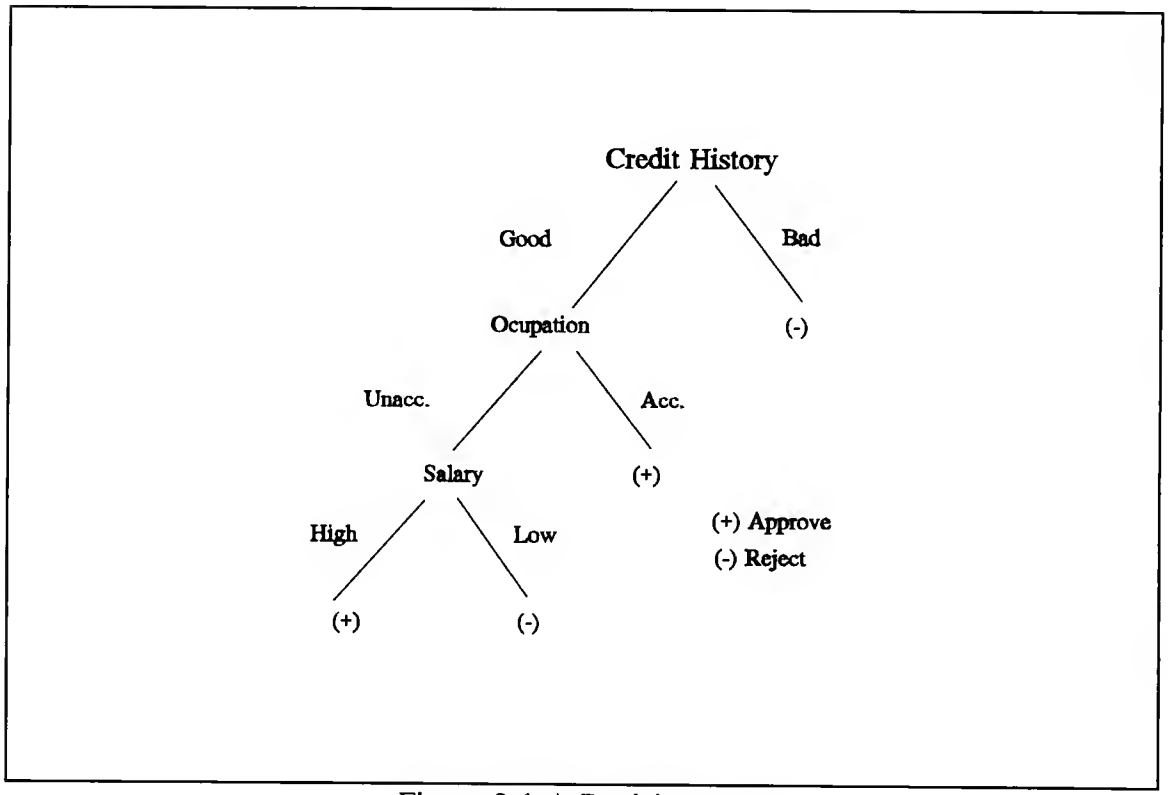

Figure 3-1 A Decision Tree

Rules are conjunctions of attributes from the node to a leaf. For example, one rule reads as IF the customer has a good credit history and an acceptable occupation THEN approve the credit.

Since the method is data driven, the resulting tree can be very different for different data sets. Also, the trees generated with the same example set may differ depending on the method used to choose attributes on which to split. Even though the accuracy of the tree is not sensitive to the split criteria used, the depth of the tree is very sensitive to this measure (Mingers 1989a). In noisy domains, where there is a lot of measurement errors, pruning the decision tree often improves the accuracy of the tree. Again the shape of the tree after pruning depends on the pruning method and there is no known optimal way of pruning (Mingers 1989b).

Decision trees have been applied to domains like medicine, agriculture, banking, production, etc. Specific applications can be found in (Messier and Hansen 1988), (Braun and Chandler 1987), (Carter and Cotlett 1987) and (Piramuthu et al. 1991).

Explanation Based Learning (EBL)--or Explanation Based Generalization (EBG)--is a knowledge intensive process. The bias required for the search is the domain theory as opposed to the inductive bias imposed by the attributes considered for concept formation by the inductive learning algorithm. In other words, the learning element has some knowledge about the domain as opposed to having no information about the environment. The task is generalization rather than acquiring new knowledge.

The mechanics of EBL are quite simple. The learning element has some knowledge about the domain it is operating in. It is given a set of examples, usually a set of small cardinality. The first task is to explain why this example is an instance of the target concept. This equivalent to theorem proving. Then, next step is to deduce results from that example. This is generalization.

For example, assume the target concept is indefinite integrals that can be taken by using integration by parts. The system has the knowledge of how to take integrals of polynomial functions, trigonometric functions, logarithmic functions, etc. The system also has the knowledge of integration techniques like integration by parts, trigonometric substitution, etc. Assume the example is $\int x^2 e^x$. The learning system must be able to explain why this is an example of integration by parts and then be able to generalize from this example that $\int x^n e^x$, $n \in \mathbb{N}$, is solvable by integration by parts.

The methodology of EBL is very close to the way humans learn through experience. For example, assume a student is learning how to take derivatives. At first, he is inexperienced, a novice. After learning the rules of taking derivatives he is able to solve problems of similar nature. However, he is still very slow and follows the steps his teacher had told him. After solving enough examples he is able skip steps, and finally he is able to find the solution without evaluating each step but rather checking a few crucial steps. At that point his knowledge is said to be tacit. Similarly, before the examples are presented the learning system is a novice, but after a few examples it is expected to generalize and draw conclusions. In other words the existing knowledge is compiled so that it is more general and in a more operational form. By this way the system not only generalizes its knowledge but also reaches to conclusions very efficiently. Many believe that this is how experts operate. Since the knowledge they have is tacit, already in the compiled form, they think and act faster and more accurately than other people in their domain of interest.

Logic Theorist (O'rorke 1989) is a system that uses EBL methods. EBL is also used in natural language understanding (Dejong and Mooney 1986), concept discovery and learning simple tasks (Mitchell, Keller and Kedar-Cabelli 1986). Research on EBL focuses on alternate strategies to improve the efficiency of learning (Flann, Dietterich 1989) . We believe that most intelligent systems will have to employ the EBL methodology for performance improvement.

Most research on EBL is restricted to very simple problems which are easily solvable by humans, like understanding a plain English paragraph or recognizing an object such as a coffee cup. Research on applying different methodologies for search and inference is still on progress.

NNs are composed of individual computing entities linked together as a network. Research on neurocomputing dates back to the 1940's and was first analyzed by Warren McCulloch and Walter Pitts (Hecht-Nielsen 1990). After an interruption for about ten years, research in this area concentrated on network architectures, and learning algorithms for NNs.

The general model of NNs has of processing elements (nodes), connections (arcs) between nodes and input and output connections. Each node has a transfer function and a local memory that consists of weights and necessary variables. Most NNs have at least three layers: an input layer, an output layer and a processing layer. The nodes in the input and output layers do not process the signals. They simply transmit it. The transfer function can give output that can effect both the local memory and the nodes' output signal. On receiving an input from an input connection the transfer function is activated

which then may activate the nodes output connection--the node fires. In this manner an input signal is transferred to the output.

Learning in NNs is accomplished by training. Training can be in supervised, or graded mode or the network can be self organizing (Hecht-Nielsen 1990). We will focus on the first two modes. In supervised training, a network is given a set of examples consisting of the pair (x_k,y_k), where y_k is the true function value. In graded mode, the network receives a grade of how well it has done on each example. In both cases each node is expected to update its local memory and to adjust its weight vector with respect to the feedback, so as to be able to discover the true function.

NNs have successfully been applied to pattern recognition, such as character recognition and many other domains ranging form credit approval to image compression (Hecht-Nielsen 1990).

The third dimension of ML research is the "nature" of learning which we classify as static, incremental and adaptive learning. In static learning, the learning system does not consider new information after the learning task has been completed. The system is ready for decision making. With incremental learning, the knowledge base is revised as new information comes in. This type of learning still assumes that the concept being learned is static. Adaptive learning can be described as a continuous cycle of generate, test and refine. The system constructs knowledge with the available information. As new information arrives, the system tests the validity of the current knowledge and refines it if necessary. Neural nets, and classifier systems are adaptive learning systems.

The fourth and last dimension we will consider is the "mode" of learning. This refers to the existence or absence of supervision during learning. We consider three levels: supervised, semi-supervised and unsupervised modes. For example, ID3 works in supervised mode since there is an ORACLE that classifies examples whereas self organizing NNs work in unsupervised mode.

Learning from Observation and Discovery, or so called "unsupervised learning", is a very general form of inductive learning where the system is not tutored by a teacher. Carbonell, Michalski and Mitchell classify systems that employ unsupervised learning by the degree of interaction with the external environment.

> The extreme points in this dimension are:
> . passive observation, where the learner classifies and taxonomizes observations of multiple aspects of the environment.
> . Active experimentation, where the learner perturbs the environment to observe the results of its perturbation. ... often this form of learning involves the generation of examples to test hypothesized or partially acquired concepts. This type of learning is exemplified in Lenat's AM and EURISKO systems (Lenat 1976, 1983.) (Carbonell, Michalski and Mitchell 1983, p. 73)

In the following section we will discuss theoretical issues on learning algorithms, and present some results of research conducted in this area. In Section 3.3 we will analyze the scheduling domain from a learning perspective and discuss the research within the four dimensional framework we have described in this section.

3.2 Theory of Learning

Until the last decade, ML research had focused on developing intelligent systems without much attention to theoretical issues like what concepts are learnable and whether

there are domains where developing a learning system is impossible. During the last decade there has been an enormous amount of research on the learnability of certain concepts, on the complexity of learning and on characterizing the learning task, especially after the pioneering work of Valiant (1984). Research has shown that some concepts are learnable by at least one learning algorithm and some are not learnable in polynomial time or require an exponentially large set of examples.

We start our discussion of these results with some definitions and notation.

Valiant proposed the idea of learning a concept with low error and doing this with high probability instead of learning a concept with complete accuracy. Later this was referred to as Probably Approximately Correct (PAC)-learning (Haussler 1987).

A learning algorithm calls a routine named EXAMPLES which selects examples with respect to a probability distribution P and labels them correctly, possibly by the help of an ORACLE. The error between the hypothesized concept and a target concept is defined as the symmetric difference between two concepts. A learning algorithm tries to generate a concept g such that the probability that the error between g and the target concept f is less than ε, i.e. $P(f \Delta g) \leq \varepsilon$, where Δ is the symmetric difference between g and f. A PAC-learning algorithm can formally be defined as follows (Natarajan 1991): An algorithm A is a PAC learning algorithm for a class of concepts F if;

a. A takes inputs $\varepsilon \in (0,1]$, $\delta \in (0,1]$ and $n \in N$ where ε and δ are the error and confidence parameters respectively, and n is the length parameter.

b. A may call EXAMPLE, which returns examples for some $f \in F$. The examples are chosen randomly according to P.

c. For all concepts $f \in F$ and all probability distributions P, A outputs a concept $g \in F$, such that with probability at least $(1-\delta)$, $P(f \Delta g) \leq \varepsilon$.

Research in this area focuses on bounds on the number of examples and the time required to learn a concept in terms of the parameters δ and ε. Most of the work done has focused on learning boolean concepts such as pure conjunctive, pure disjunctive, internal disjunctive, k-CNF and k-DNF concepts.

Assume a concept can be defined by n attributes, $a_1,...,a_n$. A pure conjunctive form is defined as the conjunction of a subset of these variables. A pure disjunctive form is defined as the disjunction of a subset of the variables. A concept is a k-CNF concept if it is represented by the conjunction of arbitrary number of clauses where a clause is a pure disjunction of at most k variables. Similarly a formula is in k-DNF form if it is represented by the disjunction of an arbitrary number of terms, where each term is a conjunction of at most k variables. An internal disjunctive concept is the same as pure conjunctive where each conjunct can be a compound atom. An attribute that is assigned a value is called an atom. If an atom is a disjunction of more than one atoms of the same attribute, it is called a compound atom.

In his pioneering work on PAC-learning, Valiant (Valiant 1984) discusses the classes of learnable concepts. He shows that certain classes of boolean concepts like k-CNF concepts, monotone DNF concepts, or arbitrary concepts with a variable appearing only once are learnable. But his findings are not generalizable to less restrictive arbitrary domains. Haussler (1988), (1989), gives bounds on the sample complexity of inductive learning which is defined as the number of examples necessary to PAC-learn a concept.

He also gives the necessary conditions for a class of concepts to be learnable with polynomial sample complexity. He reports that all pure conjunctive, disjunctive and internal disjunctive concepts are PAC-learnable. k-CNF and k-DNF concepts are also PAC-learnable with the exception that when k is large, computational efficiency is not guaranteed.

Angluin and Liard (1988) demonstrate the PAC-identifiability of CNF(k,n) concepts in noisy domains given that the noise rate is bounded above by $\frac{\varepsilon}{M+\varepsilon}$, where CNF(k,n) denotes the class of concepts in conjunctive normal form defined on n variables with at most k literals per clause, and M is the number of clauses. They also provide bounds on the number of examples needed for PAC-learnability. They conclude that in the presence of noise, PAC-learnability may not be computationally feasible for many domains.

So far we have assumed that the learning element had access to an ORACLE that labels the examples correctly. What if such an ORACLE does not exist, which is usually the case. Board and Pitt (1989) suggest such a scenario where the system tries to learn multiple concepts inductively when there is only partial information about the labels of the examples. They refer to this as semi-supervised learning. The learning element is trained with example pairs and the labeling tells if they belong to the same class or not. The authors show that most classes that are PAC-learnable such as k-CNF and k-DNF, are also semi-supervised learnable.

Most of the work done in this area is of only theoretical interest and most of the bounds on sample complexity are loose bounds with little practical value. In practice, most of the algorithms developed for learnable classes perform much better. But still there are a few open issues most of the authors mention (Valiant 1984), (Haussler 1988). Most of the algorithms assume that there is neither classification nor measurement errors on the examples. The concepts where PAC-learnability is proven are very restrictive domains. It is assumed that the attributes considered for describing the domain, the inductive bias, are enough for learning. That is, we assume there is no error due to omitting certain useful attributes. It is clear that many domains do not have these nice properties.

In the following section we will discuss the projection of these issues discussed here onto the scheduling domain.

3.3 Scheduling as a Machine Learning Problem

In Chapter 2 we characterized the scheduling problem and its domain structure. From a ML perspective, what concepts need to be learned in this domain? How could we characterize the target concept? What are the relevant attributes of the domain and the concept? Looking at the four dimensions of ML, which choices are suitable for learning in the scheduling domain?

Before we answer these questions let us briefly recall aspects of the scheduling domain. The shop floor is a highly dynamic entity where the "conditions" and "performance criteria" change frequently and there is high uncertainty. The number of

attributes required to describe the system status, which include job characteristics, machine characteristics, process characteristics, work flow characteristics etc., is very high. There is no known optimal way of doing certain tasks and thus no dependable ORACLE exists. A high ratio of classification error may be present due to lack of perfect information.

The task is learning a strategy or a set of strategies that produce good schedules. How are these strategies characterized? Are they k-DNF, k-CNF or arbitrary concepts? The answer is we do not know yet. But it is clear that we can not learn good strategies that can cover all possible scheduling environments. Scheduling environments have different characteristics and usually the concept of a good schedule varies from one environment to the other. For example, for the problems $1//C_{max}$ and $2F//C_{max}$, it is known that SPT and Johnson's rule, respectively, are optimal. Hence, we have to focus on individual environments. Even so, it is highly unlikely that one rule will be enough for classifying all good schedules within an environment. Thus we need to learn disjunctive rules that are not necessarily in any normal form. If we can characterize all relevant states of the system in a small finite number of cases, then we can learn rules that apply to each case.

Hence, learning in a scheduling environment is discovering a set of disjunctive rules that can use an arbitrary combination of the system attributes. Our purpose is to be able to discover good scheduling strategies rather than good schedules. Those strategies are simple sequencing rules that have been developed and published in literature. Basically we want to learn what sequencing rule is applicable under what conditions and

how we characterize these conditions. Fitting this view to the first dimension, we want to acquire explicit knowledge about the domain.

Analyzing the scheduling problem with respect to the second dimension requires finding the best methodology to use. This is the hardest one to justify. We will narrow our choices by explaining why we do not use certain methodologies. First, rote learning is eliminated due to its inability to generalize. Learning by analogy seems unsuitable because we can not find any domain that we can draw analogies from. Learning by being told requires an extensive use of domain knowledge which we do not have. The last choice we have is learning from examples. We will support this view because we have a large portfolio of examples, or we can generate them by simulation runs. We will also be able to characterize the relevant attributes of the system. This methodology does not require an initial domain knowledge that is expressive enough to explain the causal relationships among entities.

Next is the question of which algorithm to use. Inductive learning algorithms require the existence of an ORACLE and are highly sensitive to noise. Case based reasoning is not suitable because in such a dynamic environment seldom is a case common enough to make it worth storing. In today's competitive manufacturing environment the manufacturers are able to compete by introducing new products. This in return changes their processes and scheduling characteristics. We believe that a system will need a method that is suitable for working with imperfect information and that can conduct an efficient search.

Static learning algorithms are not suitable due to the dynamic nature of the domain. In this environment, the only system that can survive is the one that can monitor the environment and use new information to change its knowledge adaptively. Also, we not only want to acquire explicit knowledge but also improve the systems performance or at least keep it at an acceptable level in time.

Finally, due to the lack of perfect information, we believe a semi-supervised mode of learning will have to be carried out. The system will only be able to get partial information about its decisions so it must have a way of making full use of the available information. That is information given in the past and at present.

Part of the research discussed in section 2.3.4 such as Piramuthu et al. (1991), Thesen and Lei (1986) and Yih (1990), concentrate on knowledge acquisition but they do not consider rule refinement. The knowledge produced is explicit, meaning that the resulting knowledge base can explain why a certain rule applies. Lamatsch et al. (1988) skip the knowledge acquisition step and assume the existence of an initial knowledge base and employ a simple learning algorithm to further improve the performance of the existing learning system. On the other extreme, Chryssolouris (1990) use neural nets, which produce implicit knowledge. Koton's case based reasoning approach is an incremental process where the case base is generated as new examples are seen Hilliard et al. (1987), (1989) employs classifier systems to discover optimal or good rules for a known class of problems. Classifiers systems use GAs. GAs are inherently adaptive and can be used for both rule discovery and rule refinement.

A survey of GAs and classifier systems will be presented in the next chapter. We will also discuss how they can be used as learning algorithms for rule discovery and rule refinement.

All of the papers focusing on ML applications in scheduling make simplifying assumptions which limits their generalizability. Nevertheless, their results have been most encouraging.

In Table 3-1 we summarize machine learning in scheduling with respect to our four dimensional framework.

Table 3-1 Scheduling Research from an ML Perspective

AUTHOR	DEFINITION	METHODOLOGY	NATURE	MODE
Blessing and Watford (1987)	Performance Improvement	Rote Learning	Incremental	Semi-supervised
Chryssolouris, Lee and Domroese (1990)		Learning from Examples	Incremental	
Hilliard and Rangarajan (1989)	Explicit Knowledge Acquisition	Learning from Examples	Incremental	Semi-supervised
Koton (1989)	Performance Improvement	Learning from Examples	Incremental	Semi-supervised
Lamatsch et al. (1988)	Performance Improvement	Learning from Examples	Adaptive	Supervised
Nakasuka and Yoshida (1992)	Explicit Knowledge Acquisition	Learning From Examples	Incremental	Semi-supervised
Piramuthu et. al (1991)	Explicit Knowledge Acquisition	Learning from Examples (Inductive)	Static	Supervised
Thesen and Lei (1986)	Explicit Knowledge Acquisition	Learning from Examples (Inductive)	Static	Supervised
Yih (1990) and Yih (1992a)	Explicit Knowledge Acquisition	Learning from Examples (Inductive)	Static	Supervised
Yih (1992)		Learning from Examples	Static/Incremental	Supervised

CHAPTER 4
GENETIC ALGORITHMS AND CLASSIFIER SYSTEMS

4.1 Overview of Genetic Algorithms

Genetic Algorithms (GAs), first studied by Holland, are general purpose search algorithms based on an analogy to natural selection or, survival of the fittest (Goldberg 1989). Population members are represented by artificial strings, corresponding to chromosomes. The search starts with a population of randomly selected strings, and from these the next generation is created by using genetic operators. At each iteration individual strings are evaluated with respect to a performance criteria and in return assigned a fitness value, or strength. Based on their fitness values, strings are selected to construct the next generation by applying genetic operators. Even though the GA moves from one population to another stochastically, it has been shown that the algorithm "converges" (Vose and Liepens 1991a).

Compared to other optimization techniques, GAs have less restrictive assumptions. The basic differences can be summarized as follows:

> 1.GAs work with a coding of the parameter set, not the parameters themselves.
> 2. GAs search from a population of points, not a single point.
> 3. GAs use payoff (objective function) information, not derivatives or other auxiliary knowledge.
> 4. GAs use probabilistic transition rules, not deterministic rules. (Goldberg 1989a, p. 7)

GAs are also inherently parallel algorithms.

Usually the strings are defined on the binary {0,1} alphabet. For a problem that uses strings of length γ, (from bit 0 to bit γ-1), the possible number of solutions to consider is 2^{γ}. Yet, the GA is able to search this solution space very efficiently by using these three operators.

There are three operators that guide the search. As mentioned above these are probabilistic operators and do not guarantee an improvement from one iteration to another, unlike for example, the simplex method.

The first operator is reproduction. In reproduction, strings are probabilistically chosen based on their fitness values and carried to the next generation without any further modification. Strings with high fitness values have better chances of survival--survival of the fittest. This operator is applied to protect fit individuals from an accidental extinction, and possibly to increase their number in the following generations.

The second operator is crossover. Crossover is a binary operator that selects two strings based on their strength. After selecting the strings a crossover site for both strings is selected. Then the substrings to the right of the crossover site are exchanged. For example, assume S_1 = 10111 and S_2 = 11001. Assume the crossover site is chosen to be 2. After crossover the resulting strings, called offspring or children, are 10001 and 11111. Crossover helps to increase the number of similar strings in the next generation by combining the substrings (genes), of strong individuals with the hope that this will result in better individuals.

The third operator, a unary operator, is mutation. Mutation is not based on fitness values, and thus acts completely random. Mutation is carried out by randomly changing the bit values of a selected string. Assume S_1 = 10111 is chosen for mutation and assume bit 0 and 3 are probabilistically chosen for mutation. After mutation S_1 will be 11110. Mutation helps to find solutions that are not reachable by applying only crossover and reproduction. That is the only mechanism that can move a GA from a local optima when crossover does not result in radically different offspring.

Even though it is possible to implement a GA in many different ways, the simplest GA is stated as follows:

Given string size (γ), population size (n), crossover probability (χ), mutation probability (μ), a function (f) (f(x) > 0, for all x) to be optimized and a stopping criteria:

1. Set t = 0, and generate an initial population P_t of size n.
2. While the stopping criteria is not satisfied {

Find the fitness f(i) of each string i in P_t. Let

$$p(i) = \frac{f(i)}{\sum_{i=1}^{n} f(i)}$$

While all the elements of next generation P_{t+1} are not created {

Select parents S_1 and S_2 based on p(.).

Let U= U(0,1), where U(0,1) is a uniform probability distribution.

If $U < \chi$ crossover S_1 and S_2, put the offspring S_1', S_2' in P_{t+1},

else carry S_1 and S_2 to the next generation.

Select the children S_1' and S_2'

For (i = 0 to γ-1) {

if U= U(0,1) < μ,

flip the i^{th} bit

}

}

Set t = t + 1.

}

3. The resulting population is the solution.

Due to their stochastic nature GAs do not guarantee an optimal solution at their termination. However, empirical experience has shown that they converge to good quality, near optimal solutions very quickly. In the next section we will explain this behavior of GAs.

4.1.1 The Fundamental Theory of GAs and Implicit Parallelism

Schemata theory (Goldberg 1989) and implicit parallelism (Goldberg 1989), concept are useful in explaining why the GAs quickly converge to a high fitness population of strings.

Consider a string of length γ defined on the alphabet {0,1} and let $S=\{0,1\}^{\gamma}$. Then a schema H defined on this alphabet is a subset of S such that the strings in S have the

same values at the same bit positions for some bit positions. For example, assume γ=4, and assume we fix the values at the 1st and 3rd bit position then

H={0110, 0010, 0111, 0011}.

H can also be represented as a string using the alphabet {0,1,*} where * means either a 0 or 1. In our example H is 0*1*.

The length of a schema H, $\delta(H)$, is the distance between the first and the last fixed positions. The order of a schema , o(H), is the number of fixed positions (Goldberg 1989). In our example, $\delta(H) = 3-1 = 2$ and $o(H) = 2$.

The fundamental theorem of GAs states that "short, low order above-average schemata receive exponentially increasing trials in subsequent generations," (Goldberg 1989), pp. 33. The expected number of members of H in the next generation, m(H,t+1) is bounded by,

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left[1-\chi\frac{\delta(H)}{\gamma-1}-o(H)\mu\right]. \qquad (1)$$

The term f(H) is the average schema fitness, whereas $\bar{f}$ is the average population fitness. χ and μ are the crossover and mutation rates respectively. The reproduction operator tries to preserve the schema whereas the crossover and mutation operators disrupt the schema. That is why short, low order schemata have better chances of survival. Such low order, short schemata are called building blocks since they lead to better solutions. So, after a few iterations, the number of strings belonging to such useful schemata will increase rapidly. For example, assume $f(x) = x^3$, $\gamma = 3$, H_1= 1**, and H_2=1*0. H_1 has better

chances of survival because $f(H_1)=187$, $f(H_2)=140$ and H_1 is a low order schema with short defining length. It will be less prone to disruptions by crossover and mutation.

One other reason why GAs converge to a population composed of strings with high fitness is explained by implicit parallelism. Most optimization techniques search by separately considering promising solution points and use a bookkeeping mechanism to remember such solutions. For example, the Branch and Bound algorithm keeps the best solution and the lower and upper bounds associated with each node. As the size of the search tree increases the bookkeeping effort also increases. GAs, however, do not have such an explicit bookkeeping mechanism and yet at each iteration they are able to search a population of points. Moreover, Holland has estimated that they actually process $O(n^3)$ schemata without any additional bookkeeping (Goldberg 1989a). That is why they can quickly eliminate a family of solutions that are not promising without explicitly considering them, and thus, "converge" to a solution very rapidly.

4.1.2. An Exact Representation of a GA's Search Behavior

The schemata theorem explains why GAs are expected to stochastically converge to a final population. It simply states that as time passes similar strings that are above average fitness will increase in number. The expression for the number of schemata in the next generation given in (1) is only an upper bound. Moreover, it does not tell us anything about the individual strings in the next generation. In a series of papers Liepins, Vose, and Nix attack this issue and give an exact representation of the GA behavior.

Liepins and Vose (1992) show that crossover can be represented by using the bitwise multiplication, $\otimes$, and exclusive or, $\oplus$, operators with a string m, referred to as a mask whose use will be clear soon.

Throughout this chapter and chapter 5 we will use the notation introduced below. We will refer to binary strings by the integers they represent. Frequently, indexes will mean population members. Let,

γ be the string length.

N be the total number of strings that we can generate with γ bits, i.e, $N=2^{\gamma}$.

τ be a binary string of length γ, with all its bits equal to 1.

Ω be the set of all binary strings of length γ representing integers from 0, ..., N-1.

$r_{i,j}(k)$ be the probability of obtaining k as an offspring when combining parents $i, j \in \Omega$.

n be the population size.

t be the iteration or population index.

p^t be a vector of size N where p_i^t corresponds to the proportion of string i in population t.

s^t be a vector of size N where s_i^t is the probability of selecting i for recombination in population t.

F be an NxN diagonal matrix, where $F_{i,i} = f(i)$, $f(i) > 0$, for i = 0..N-1 and f is the function of to be maximized.

M be an NxN matrix, where $m_{i,j} = r_{i,j}(0)$.

M_* be an NxN matrix, where $(m_*)_{i,j} = m_{i \oplus j,i}$.

$E[x]$ be the expected value of a random variable x.

$|x|$ be a norm on $x \in \mathbb{R}^N$ defined as $\sum_{i=0}^{N-1} x_i$.

Definition 4.1 (Liepins and Vose 1992): For any $m \in \Omega$, $\bar{m} = m \oplus \mathbf{1}$.

Definition 4.2 (Liepins and Vose 1992): X is a crossover operator if there exists an $m \in \Omega$ such that $X(x,y) = (m \otimes x) \oplus (\bar{m} \otimes x)$.

Theorem 4.1 (Liepins and Vose 1992):Let X be a crossover {operator} as defined above. Then X commutes with addition and projection and preserves schemata. Conversely, let X be a binary recombination operator that preserves schemata and commutes with addition and projection. Then X is a crossover operator.

Theorem 4.2 (Liepins and Vose 1992) Let X be any arbitrary fixed crossover operator and $x, y \in \Omega$. Let X(x,y)=(u,v). Then X(u,v)=(x,y). Moreover if X_m and $X_{\bar{m}}$ are crossover operators with masks m and $\bar{m}$ respectively, then for any $x, y \in \Omega$, $X_m(x,y) = X_{\bar{m}}(y,x)$.

Vose and Liepins (1991a), further characterize the GA behavior by deriving exact probabilities for the existence of an individual in the next generation given two strings x and y are selected for recombination. They also show that the iterates of a GA with infinite population size can be modeled as the iterates of a nonlinear operator.

Below, Lemma 4.1 characterizes the expected proportions of each string k in the next population.

Lemma 4.1 (Vose and Liepins 1991a):

$$E[p_k^{t+1}] = \sum_{i=0}^{N-1} \sum_{i=0}^{N-1} s_i^t s_i^t r_{i,j}(k).$$

Lemma 4.2 (Vose and Liepins 1991a): If recombination is a combination of mutation and crossover then

$$r_{i,j}(k \oplus l) = r_{i \oplus k, j \oplus k}(l).$$

Lemma 4.2 states that probability of producing a child q that can be represented by $k \oplus l$ is equal to producing one of the operands when the parents are shifted dyadicly by the other child. In other words, the probability of producing a child is invariant under a dyadic translation. Lemma 4.2 does not say that both sides of the equality will result in the same offspring, but simply states that the probabilities are the same.

For $s \in \mathbb{R}^N$, let s^T denote the transpose of s. Let σ_j is a permutation defined on $\mathbb{R}^N$, by

$$\sigma_j s = \sigma_j <s_0,....,s_{N-1}>^T = <s_{0 \oplus j},...,s_{N-1 \oplus j}>.$$

Define $\bar{M}$ as

$$\bar{M}(s) = <(\sigma_0 s)^T M(\sigma_0 s),...,(\sigma_{N-1} s)^T M(\sigma_{N-1} s)>^T.$$

Definition 4.3 For all $x, y \in \mathbb{R}^N$, $x \sim y$ if and only if for some $\lambda > 0$, $x = \lambda y$.

Theorem 4.3 (Vose and Liepins 1991a): $E[s^{t+1}] \sim F\bar{M}(s^t)$.

We will give an obvious extension of Theorem 4.3 in Lemma 4.3.

Lemma 4.3: The expected probabilities in the next generation are given by,

$$E[s^{t+1}] = \frac{F\bar{M}(s^t)}{|F\bar{M}(s^t)|}.$$

Vose and Liepins (1991a), further characterize the behavior the F and M matrices. They show that the basin of attraction of a fixed point of F is the intersection of the unit sphere S and the ellipsoid given by,

$$\sum_{i=0}^{N-1} \left(s_i \frac{f(i)}{f(j)} \right)^2 < 1.$$

They further discuss the fixed points of operator $\bar{M}$, and state the following theorem.

Theorem 4.2 (Vose and Liepins 1991a): Let $x \in \bar{M}_{fixed}$, {where $\bar{M}_{fixed}$ is the set of fixed points of operator .} If the matrix M is positive, then x is asymptotically stable whenever the second largest eigenvalue of M_* less than $\frac{1}{2}$.

They further conjecture that the second largest value of M_* is $\frac{1}{2} - \mu$ and the third largest eigenvalue of M_* is

$$2\left(1 - \frac{\chi}{\gamma - 1}\right)\left(\frac{1}{2} - \mu\right)^2.$$

Later it was shown by Koehler (1992) that the full spectrum of M_* can be calculated by

$$\frac{(1-2\mu)^{|i|}}{2}\,\frac{1-\chi^{wid(i)}}{\gamma-1},\ i=0,\ldots,2^{\gamma-1}$$

where wid(i) is the number of bits strictly between the highest and the lowest non-zero bits of i. This result shows that the operator $\bar{M}$ has a stable fixed point.

Finally the authors explain the expected behavior of GAs. The population proportions move to a fixed point of $\bar{M}$ under the influence of F, in contrast to a genetic drift, and will jump out of that point's basin of attraction if that point is not maximally fit, to be trapped in the basin of attraction of another fixed point. This clearly explains why GAs first slow down in improving the solution and then suddenly jump to another solution.

Parallel to this research, Davis (1991), analyzed the behavior of GAs from another point of view. He modelled each population as a state of a Markov Chain (MC), and he also gave expressions for exact transition probabilities for reproduction only, for reproduction and crossover, and finally for reproduction, crossover and mutation. He assumed the mutation rate to be varying and derived ergodicity bounds on the mutation rate.

Goldberg and Segrest (1987) also discuss GA behavior by modeling it as MC. Their model is far from representing the real behavior of GAs, since they consider a population composed of one bit strings. They analyze genetic drift and convergence under reproduction. They use the first passage times to analyze the expected time to different levels of convergence.

Vose and Liepins (1991a), derive an exact probability expression for the probability of generating 0 by recombination on i and j. They show that,

$$r_{i,j}(0) = \frac{(1-\mu)^{\gamma}}{2}\left[\eta^{|i|}(1-\chi+\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{-\Delta_{i,j,k}})+\eta^{|j|}(1-\chi+\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{\Delta_{i,j,k}})\right]$$

where

$$\eta = \frac{\mu}{1-\mu}$$

and

$$\Delta = |(2^k-1)\otimes i| - |(2^k-1)\otimes j|.$$

Before we introduce the Markov Chain model let us introduce more notation. Let

Z be an $N \times |P|$ matrix, where each column refers to a population with $z_{i,j}$ being equal to the number of string i in population j.

Φ_i represent population i, or equivalently the corresponding column of Z.

P be the set of all populations.

Q be an $|P| \times |P|$ Markov state transition matrix.

The number of all possible combinations of populations for a given γ and n is

$$|P| = \binom{2^{\gamma}+n-1}{2^{\gamma}-1} = \binom{N+n-1}{N-1}.$$

A MC defined on these states will have $|P|$ number of states, each defined by the a column of matrix Z. Following Nix and Vose (1992), we will show how the exact transition probabilities are calculated.

For example let $\gamma = 2$ and $n = 2$. Then $\Omega = \{00, 01, 10, 11\}$ and P = { {00, 00}, {01, 01}, {10, 10}, {11, 11}, {00, 01}, {00, 10}, {00, 11}, {01, 10}, {01, 11}, {10, 11} }. Then the incidence matrix Z is

$$Z = \begin{bmatrix} 2 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Nix and Vose (1992), showed that the transition probabilities from state, or population, i to j are multinomially distributed and are given by,

$$Q_{i,j} = n! \prod_{y=0}^{N-1} \frac{\left(\frac{F\phi_i}{|F\phi_i|} \right)^{\gamma_{y,j}}}{z_{y,j}!}.$$

The authors further showed that as the population size increases the iterates of the MC converge to the iterates of the operator

$$G(s) = \bar{M}\left(\frac{F\phi_i}{|F\phi_i|} \right).$$

They also showed that since the mutation rate is non-zero Q is ergodic, and a limiting steady state distribution π exists. They also proved that as n tends to infinity the limiting distribution π converges to π^* where

$$\pi^* = \lim_{n \to \infty} \pi .$$

Moreover, π^* will have positive values for elements that correspond to the fixed points of G. That, is for large population sizes the iterates of MC converge to the iterates of operator G.

4.1.3. Current Research On GAs

GA Research can be summarized into two categories: research using GAs as search algorithms in an application domain and research on issues about GAs, such as deceptiveness, long term behavior, short term behavior, convergence, coding etc. In this section we will discuss both with the emphasis on scheduling applications and the search behavior of GAs.

As explained earlier, deceptiveness is a crucial issue in GA optimization. Researchers have shown ways of characterizing deceptive functions and failure modes. Liepins and Vose (1990) define four modes of failures: inappropriate embedding, sampling error, crossover disruption and deceptiveness.

In problems where there are a lot of very close peaks and valleys, a high resolution of the real domain is required. The authors suggest dynamic embedding to overcome such difficulties. For example, the population can be monitored and when it converges to points very close to each other, the resolution can be increased and thus more points in that neighborhood can be searched (Liepens and Vose 1990).

Sampling error occurs when the GA can not estimate the utility, the average fitness, of the schema and focuses on other inferior schemata. Increasing sampling size, randomizing the initial population and keeping only one copy of each individual are suggested solutions (Liepens and Vose 1990).

Crossover disruption is a threat to the GA when the current population has long defining length schemata. Such schemata are more susceptible to extinction due to crossover. Assume, the optimal solution for a 5 bit problem is 11111 and assume that the current population contains schemata such as 1***1, and **00*. Even though the first schema contains the optimal solution it is very unlikely to survive because of crossover. One remedy for this case is using an inversion operator (Liepens and Vose 1990).

Some functions are known to be deceptive for GAs. Functions, where low order schemata lead to suboptimal solutions are deceptive functions and GAs have difficulty with finding the optimum of such functions. It is analogous to the situation where the estimate of the gradient of the function leads the search to a suboptimal point. Liepens and Vose (1991b) show that it is possible to implement deceptive functions as a combination of an easy function and a spike function. They also show that it is also possible to convert deceptive functions to easy functions by affine transformations. However, under the new mapping there may be another function that is deceptive.

Goldberg (1989b), (1989c) explains how to analyze deceptiveness, by using Walsh transforms. Walsh transforms first suggested by Bethke (Goldberg 1989a) are useful particularly in calculating average schemata fitnesses. Goldberg also gives an algorithm, ANODE to analyze deceptiveness. One drawback of the algorithm is that it requires the

optimal value of the function. In a real-life optimization problem, the optimal value is almost never available.

Goldberg, Korb and Deb (1989), (1990) analyze a new class of GAs called messy GAs (mGAs). mGAs work on variable length strings, where the crossover operator is replaced with cut and splice operators. One substantial difference from the classical GA is that the initial population is generated by enumerative methods to ensure that there is at least one copy of the building blocks of specified size. An mGA works in two phases. In the "primordial" phase, the building blocks that will generate good solutions when combined are selected and the population size is reduced. After this phase the cut and splice operators are applied which constitutes the "juxtapositional" phase. The authors present a schema analysis of the mGA and report successful results on deceptive functions.

4.1.4. GA Applications on Scheduling

GA applications on scheduling focus on the Traveling Salesman Problem (TSP) because of its high coupling with the job-shop problem. TSP is a well studied NP-complete problem (Gary and Johnson 1979). The problem can be stated as follows: Given a graph G(V,E) and related arc costs, find a minimum cost simple tour such that each node is visited only once.

All GA approaches to TSP focus on coding the problem such that a particular string represents a feasible tour, i.e., the bits of the string are ordered. There are numerous operators developed and tested that guarantee feasible tours in the next

generation. One such operator is described in Whitley, Starkweather and Fuquay (1989). The "edge recombination operator" is designed to preserve the common edges appearing in both parents. The edge recombination operator does not need any local information such as the cost of the edges, but rather works with global information. The operator has been successfully applied to the TSP problem as well as to the job-shop problem.

Cleveland and Smith (1989), describe how GAs could be used for the sector scheduling problem. The authors first show that the original problem is equivalent to a one machine problem and then proceed to show its link with TSP. They compare the relative performances of different crossover operators and some known heuristic rules on the sector scheduling problem. Their findings are in favor of using a subtour chunking operator against the other operators and heuristic rules. They also found that using domain specific knowledge significantly improved the performance of the operators. Despite these results, they conclude that while considering more realistic functions such as WIP costs, pure sequencing approaches are not suitable.

An interesting application of GAs to scheduling was given by Husbands, Mill and Warrington (1991). The authors consider the process planning and the scheduling problem as a whole and use GAs to construct process plans helpful to construct good schedules. A process plan is represented by a string suitable for crossover, reproduction and mutation. These strings are evaluated within a simulation and the actual scheduling decisions are made independently if there are no conflicts between plans. If a conflict occurs, according to one scheme, a more fit plan gets the resource. The authors also tried using "arbitrators whose genetic code included which population had precedence at any

stage." The strength of arbitrators are updated throughout the simulation. As a result, the system discovered not only good planning-scheduling strategies but also learned effective conflict resolution plans. The system is reported to reduce both machining costs and total elapsed time.

4.2. GA Based Learning Systems and Classifier Systems

GA Based Learning Systems (GABLSs) are usually hybrid systems that use GAs for discovering useful rules for the domain of interest. One family of GA learning systems are Classifier Systems (CSs). CSs are distinguished by the way they use the rules and by their conflict resolution and reward mechanisms.

Classifier Systems (CSs), first proposed by Holland (Goldberg 1989a), are learning systems that are composed of three parts: a rule and message system, a credit assignment algorithm, and a genetic algorithm.

A rule and message system enables the system to store knowledge and communicate with the external and internal information. The detectors enable the system to receive information from the outside world which is put into the message list. Any rule that matches this information is invoked, which, in turn, puts its own message in the message list or activates an effector. Rules are simple codings of production rules usually defined on the alphabet {0,1,#}, where # means do not care or match everything. If the premise of the rule matches one of the messages in the message list, it fires, and thus, posts its own message to the message list or to the external environment. Messages are usually defined on the alphabet {0,1}.

Each classifier is assigned an initial strength. Based on its message, the system receives a reward from the external world. A credit assignment algorithm is used to determine how to distribute this reward back to the rules that took part in posting the output message. One such algorithm is the Bucket Brigade (BB) algorithm, proposed by Holland (Goldberg 1989a). BB simulates a market economy where each rule that matches the current message makes a bid based on their fitness values. Those who bid then post their message on the message list and pay the bid amount to the owner of the message they had matched. This is carried out until one classifier posts an output message and receives a reward. In this algorithm the external reward is given only to the owner of the output message. The strength of classifier i at time t is given as

$$S_i(t+1) = S_i(t) - P_i(t) - T_i(t) + R_i(t).$$

The term $P_i(t)$ is the bid amount, $T_i(t)$ is the tax amount, and $R_i(t)$ is the reward classifier i receives--reward may be zero. This is not the only way of distributing the reward back. We will discuss other methods later in this section.

Due to the increase in the computational complexity when multiple rules are fired all knowledge based systems need a mechanism to break ties and select a rule to apply among those competing. Most CSs use a competition based conflict resolution. One conflict resolution mechanism is called "noisy bidding" where the bid made by the classifier is $C_{bid}S_i + e$, where $e \sim N(0,\sigma_{bid})$. Only a subset of the matching classifiers are selected to be fired based on their bid values (Goldberg 1989a). One other scheme uses

the $C_{bid}S_i$ values to construct a probability distribution, (a roulette wheel), which is used to select only a subset of the classifiers.

If a CS had only the classifiers and the credit assignment algorithm, it would be able to discover rules that are useful only among those that initially are given to the system. What if there are other useful rules that would result in improved performance? Without a capability to discover new rules the system can not improve any further. A GA is employed to overcome this problem which works as described in Section 4.1 with the possible exception that the strings are not necessarily integers or reals but elements of an arbitrary set. Classifiers are chosen for reproduction and crossover based on their fitness values and again mutation is applied randomly. Since CSs try to prevent drastic changes in the rule base to promote stability, crossover and mutation rates are lower when compared to GAs used for optimization.

Research on CSs focuses on the credit assignment algorithms and bidding schemes employed for conflict resolution, as well as on some practical applications of CSs to different domains. Versions of CSs have been generated and used for many different domains like medical diagnosis, finding rules to predict company profitability, scheduling, etc., (Goldberg 1989a).

Grefenstette (1989) describe a system, SAMUEL, that utilizes GAs for rule discovery. Conflict resolution is based on a utility measure--not described in the paper--associated with the active rules. All active rules have their strengths adjusted incrementally at the end of each episode. The author utilizes both the mean strength and the variance to represent the current strength of the rule. After a few episodes the rules

are updated by using genetic operators. The author reports fast convergence to useful rules.

GA use is wide spread. Greene and Smith (1992) use GAs in connection with a filtering mechanism to produce good rules. At each iteration, the system, COGIN, uses a set of examples for filtering the weak rules. One uncommon feature of this approach is that the system does not maintain a fixed population size. At each iteration, some rules are paired for reproduction. The resulting rules are combined with the old population and tested against a set of examples. Those which can classify the examples are carried to the next generation and the rest is discarded. The system is reported to perform as well as a decision tree induction method, C4, which is a descendant of ID3 (Quinlan 1986).

Some research concentrates on comparing the performances of credit assignment and conflict resolution algorithms. Grefenstette (1988), compare BB and the Profit Sharing Plan (PSP) algorithms. PSP distributes the external reward to all the rules that helped to get this reward whereas in BB only the last rule in the chain gets the external reward. The strength update in PSP is explained by

$$S_i(t+1) = S_i(t) - b\,S_i(t) + b\,p(t).$$

The author shows that PSP is better in predicting the external rewards, that is, the steady state strength of rules, compared to BB. The author further suggests that BB is a subgoal reward scheme that tends to even out the strengths of the rules in a given chain. Furthermore, BB is more suitable for parallel rule firings than PSP. Liepins et al. (1991) compares the steady state behavior of Backwards Averaging (BA), BB and PSP. BA and

PSP use the same credit assignment scheme. The only difference is BA reduces the reward by a constant factor but, PSP does not. In BA, assuming there is a chain of n-1 classifiers the strength is updated by,

$$S_i(t+1) = S_i(t) - bS_i(t) + b\lambda^{n-i}p$$

for $0 < \lambda < 1$. They show that BB and BA have the same steady state strengths for a linear chain, when BB uses fixed percentage tax scheme. They also report that BA does not enforce the development of short chain of rules. Zhou (1990) discusses and demonstrates the usefulness of a long term memory for a classifier system.

Goldberg (1990) discusses a very interesting problem encountered in decision making. The problem is as follows: Given information about the outcomes of two events, find a decision rule to minimize the number of incorrect guesses of the outcome at each trial. Even though the optimal strategy is known to be to always select the outcome that has the highest probability of occurring, it is observed that humans behave in a suboptimal way. They tend to base their decisions on the probability distribution of the outcomes. This is referred to as Probability Matching (PM). The author shows that roulette wheel bidding is equivalent to PM and noisy bidding can be modified to mimic both PM and the optimal decisions. He also proposes bidding scheme, variance-sensitive bidding, where $B_j = E[B_j] + \sigma(B_j)$. Finally the author shows that variance-sensitive bidding is more flexible in handling changes in the information. For example, this handles the case where the probability of each outcome is reversed.

CSs have been applied in almost every domain but Liepins and Wang (1992) show that there are domains where they are not suitable. They give a sketch of their ideas as to why GAs are not suitable for stationary concepts. They also state that a CS is "uniquely well suited" for non-static domains. Their results are in agreement with Goldberg's (1990). Goldberg concludes that CSs have an inherent mechanism to handle changing information patterns since they utilize probability matching but may be suboptimal for static environments.

The findings by Goldberg, and Liepins and Wang (1992) suggest that the CSs are suitable for domains where there is a lot of uncertainty and possible noise in the incoming data. The CSs are robust enough to handle such cases.

Wilson (1987), describes the animat problem. According to his description, an animal

> . is born with some associations, but the rest of the knowledge necessary must be learned through experience;
>
> . has to learn incrementally;
>
> . "does not have the luxury of a teacher-like environment to tell it which action is right in each situation or even to provide an error signal. Instead, on those relatively rare occasions when action brings immediate satisfaction {payoff} of a current need the best the animal can do is rate the action (and the responsible rule) according to the degree of satisfaction." (Wilson 1987, p. 200);
>
> . has to learn under payoff that may not be available right after action;
>
> . should have the capability to disregard the irrelevant attributes of the environment and generalize its findings as much as possible.

He further claims that an animat should learn "disjunctive" concepts. He reports successful application of CSs to the animat problem.

The animat problem has many resemblances to the scheduling problem. The learning system has some domain knowledge but does not know the causal relationships between actions and system attributes. Furthermore, there is no supervisor that can tell the system what to do. It has to make its own decisions on some performance measure supplied by the management. Moreover the payoff it receives does not explicitly say how well it performed and it can contain unnecessary or missing information (operationalization of the performance measure) and it can be continuously changing in time. The system has to understand the relative merit of each rule (rule usefulness discovery). Finally, the system has to use its past knowledge to direct its search for new knowledge.

Even though we do not have a justification yet, based on the results of Goldberg (1990), Wilson (1987), and Liepins and Wang (1991) we believe that a GA based learning approach is most suitable for the scheduling domain.

In the next chapter we will discuss some technical issues of GAs and in Chapter 6 we will show how a GA based learning system can be used for performance improvement as well discovering useful dispatching rules in scheduling.

CHAPTER 5
ISSUES ON FINITE GAs

5.1 An Issue on Stopping Criteria

To date, all convergence properties of GAs have been empirically studied with the general consensus that they "converge." However, GAs are stochastic algorithms which use probabilistic "pivoting" rules in contrast to deterministic pivoting rules of, say, the simplex method. Hence it is not possible to talk about convergence to a point. Nonetheless, we can talk about a steady state behavior of a GA. Nix and Vose (1992) showed that a steady state behavior exists and that it is not possible for a GA to be trapped at a particular inferior solution indefinitely. The probability of visiting a good solution (including the optimal) is considerably higher than other solutions. This point was first discussed by Vose and Liepins (1991a). They showed that a GA will be trapped at the basin of attraction of a good solution but will, with high probability, eventually move out to be trapped in the basin of attraction of a better solution. Hence, it is not safe to use stopping rules like "stop when there is no significant improvement during the last ten iterations" since we may be temporarily stuck in the basin of attraction of an inferior point.

In this chapter we develop a bound on the number of iterations required to achieve a level of confidence that guarantees that the GA has found an optimal solution. We restrict our attention to GAs operating on finite populations with integer members (i.e.,

the search space is composed of integer points). Our model is based on work by Nix and Vose (1992), Davis (1991), Vose and Liepins (1991a) and the concept of first passage time distributions of Markov Chains.

In Section 5.2 we discuss some preliminaries, develop our model and derive our exact bound. We also present a bound sufficient to guarantee a desired confidence level and is easy to compute. In Section 5.3 we discuss some properties of the bound.

5.2 Stopping Criteria for GAs

In this section we will use the model developed by Nix and Vose (1992) and Davis (1991), and derive an upper bound on the number of iterations necessary to assure seeing an optimal solution with a specified confidence level by using the concept of first passage times of a Markov Chain. First we discuss some preliminaries.

5.2.1 First Passage Times: Preliminaries

First passage times are used to explain the transition behavior of a Markov Chain between states. For example, a marketing research study designed to analyze the buying behavior of consumers may be modelled as a MC. The first passage times may then be used to explain when a consumer will first switch from product A to product B. While analyzing the behavior of a GA, the first time a GA discovers an optimal solution starting from any state is of particular interest. Let,

X_i denote the state the MC is in at time i.

P be the set of all states of the MC.

Q be the Markov state transition matrix for our GA. We assume that the mutation rate is positive so that Q is ergodic. We also assume that the mutation rate is less than 0.5.

$Q_i\{X_t=j\}$ be the probability that the MC is at state j at time t starting from state i.

T_i be the i^{th} time the MC has visited state j. T_1 is the time of the first visit to state j.

$\Phi_k(i,j)$ be the probability that it takes the MC k transitions to visit state j starting at state i.

$\phi_k(i)$ be equal to $\Phi_k(i,j)$ for all $i \in E$, and for some fixed j.

ϕ_k be a $|P|$ x 1 vector where $\phi_k(i)$ is the probability that it takes the MC k transitions to first enter state j starting at state i for some fixed j.

e_j be a Nx1 unit vector with zeros everywhere except for the j^{th} position.

I be the $|P| \times |P|$ identity matrix.

e be a vector of ones.

$\rho(A)$ be the spectral radius of matrix A.

We can derive $\Phi_k(i,j)$ for two cases. For $k = 1$,

$$\Phi_k(i,j) = \text{Prob}_i\{T_1 = 1\} = \text{Prob}_i\{X_1 = j\} = Q(i,j), \qquad (1)$$

and for $k \geq 2$,

$$\Phi_k(i,j) = \sum_{b \in E-\{j\}} \text{Prob}_i\{X_1 = b\}\,\text{Prob}_b\{X_1 \neq j, \ldots, X_{k-2} \neq j, X_{k-1} = j\}. \qquad (2)$$

Combining (1) and (2) we have,

$$\Phi_k(i,j) = \begin{cases} Q(i,j) & k=1, \\ \sum_{b \in E-\{j\}} Q(i,b)\Phi_{k-1}(b,j) & k \geq 2. \end{cases} \quad (3)$$

Finally Equation (3) can be stated in matrix form for fixed j by,

$$\phi_k = (Q - Qe_j e_j^T)\phi_{k-1}, \text{ for } k \geq 2,$$

or equivalently

$$\phi_k = (Q - Qe_j e_j^T)^{k-1} Qe_j, \text{ for } k \geq 1.$$

5.2.2 An Upper Bound on the Number of Iterations

Our problem can be defined as follows: Given a desired confidence level δ, the population size and the string length how many iterations does a GA require before we can stop so that we will have seen an optimal solution with at least δ probability. That is, find the least t such that,

$$\text{Prob}\{\text{An optimal solution has been found} \mid \text{The GA had at least t iterations}\} \geq \delta$$

Our problem, can be equivalently formulated as,

(Problem 1) $\max_j \{\min t\}$

subject to

$$\sum_{i=1}^{t} (Q - Qe_j e_j^T)^{i-1} Qe_j \geq \delta e. \quad (4)$$

Before we proceed to solve Problem 1, it is noteworthy to give some basic results on the sum of the power of matrices and the spectral properties of non-negative matrices.

The following trivial equality holds for any matrix A when $(I-A)^{-1}$ exists:

$$\sum_{i=1}^{t} A^{i-1} = (I - A^{t})(I-A)^{-1} \tag{5}$$

Theorem 5.1 (Seneta 1973): (The Subinvariance Theorem). Let T be a non-negative irreducible matrix, s a positive number, and $y \geq 0$, $y \neq 0$ a vector satisfying

$$Ty \leq sy.$$

Then (a) $y > 0$ and (b) $s \geq r$, where $r = \rho(T)$ is the Perron-Frobenius eigenvalue of T. Moreover, $s = r$ if and only if $Ty = ry$.

Our first result is given below.

Theorem 5.2: The minimal number of iterations required to guarantee that we have seen the optimal solution with probability at least δ is

$$t^{*} = \max_{j} \left\{ \frac{\ln(1-\delta)}{\ln \rho(Q - Qe_j e_j^{T})} \right\}.$$

Proof: Consider

$$\sum_{i=1}^{t} (Q - Qe_j e_j)^{i-1} Qe_j. \tag{6}$$

By using (5) we can write (6) as

$$[I - (Q - Qe_j e_j^{T})^{t}](I - Q + Qe_j e_j^{T})^{-1} Qe_j$$

To use (5) we need to show that $(I-Q+Qe_j e_j^{T})$ is non-singular. To see this, consider the characteristic equation of $(Q-Qe_j e_j^{T})$ which is equal to $\det(\lambda I - Q + Qe_j e_j^{T}) = 0$.

We know that the solution set of this characteristic equation does not include $\lambda = 1$, because the row sums of $(Q - Qe_je_j^T)$ are strictly less than 1 and, thus, the maximal eigenvalue can not be 1, (see Theorem 3.3 below.) Hence $(I - Q + Qe_je_j^T)$ can not be singular and thus, its inverse exists.

Claim 1.

$$(I - Q + Qe_je_j^T)^{-1} Qe_j = e \tag{7}$$

Proof of Claim 1: Assume the claim is not correct. Multiply both sides of (7) by

$$(I - Q + Qe_je_j^T).$$

The right side gives $e - e + Qe_j$ since $Qe = e$. Hence we see that the assumption implies

$$Qe_j \neq Qe_j,$$

a contradiction. Thus the claim is true.

Hence (4) can be rewritten as

$$\left[I - (Q - Qe_je_j^T)^t\right]e \geq \delta e. \tag{8}$$

We now show using Theorem 5.1 that, satisfying (8) is equivalent to satisfying

$$\rho\left[(Q - Qe_je_j^T)^t\right] \leq (1 - \delta). \tag{9}$$

We can not directly apply Theorem 3.1 to $(Q - Qe_je_j^T)$ since it is not irreducible. However, $(Q - Qe_je_j^T)$ has the same eigenvalues as the matrix $\bar{Q}_j$, where $\bar{Q}_j$ is the $| P-1 | \times | P-1 |$ matrix obtained by deleting the j^{th} column and row of matrix Q. To see this, consider the characteristic equation of $(Q - Qe_je_j^T)$, $\det (I - Q + Qe_je_j^T) = 0$. Notice

that the matrix ($Q - Qe_je_j^T$) has all zeros in its j^{th} column. Then the characteristic equation will be of the form $\lambda(...)(..)....(..) = 0$. But $\bar{Q}_j$ has the same elements of ($Q - Qe_je_j^T$) except for the j^{th} column and the j^{th} row elements, thus the only difference between the characteristic equations of the two is the first term λ, which must be equal to zero since the characteristic equation is of the form $\lambda(...)(..)....(..) = 0$. Finally, we know that since the mutation rate is non zero all states of $\bar{Q}_j$ communicate with each other. Hence, it is irreducible and we can safely use Theorem 3.1 on $\bar{Q}_j$ and, thus, obtain results for ($Q - Qe_je_j^T$).

Hence, by Theorem 3.1 we can rewrite (9) as

$$\rho[(Q - Qe_je_j^T)^t] \leq (1-\delta).$$

But $\rho(A^t) = \rho^t(A)$. Thus we can write

$$\rho^t\left[(Q - Qe_je_j^T)\right] \leq (1-\delta).$$

By taking the logarithms of both sides we get

$$t\ln\left[\rho(Q - Qe_je_j^T)\right] \leq \ln(1-\delta).$$

Rearranging terms gives

$$t \geq \frac{\ln(1-\delta)}{\ln\rho(Q - Qe_je_j^T)}.$$

Now we can restate Problem 1 as,

$$\max_j \{ \min t \}$$

subject to,

$$\frac{\ln(1-\delta)}{\ln\rho(Q-Qe_je_j^T)} \leq t.$$

Problem 1 has a solution t^* given by

$$t^* = \max_j \left\{ \frac{\ln(1-\delta)}{\ln\rho(Q-Qe_je_j^T)} \right\},$$

which proves the theorem. Q.E.D.

Determining the spectral radius for each $(Q - Qe_je_j^T)$ is not easy. However, we can find an upper bound for each spectral radius and choose the maximum to provide an upper bound on t.

Let α_j be any number such that

$$\rho(Q-Qe_je_j^T) \leq \alpha_j < 1,$$

and let

$$\bar{t}_j = \frac{\ln(1-\delta)}{\ln\alpha_j}.$$

Then for

$$\bar{t} = \max_j \bar{t}_j,$$

we will have

$$\bar{t} \geq t^*.$$

The next result gives us one way to choose α_j values.

Theorem 5.3 (Minc 1988) : If A is a non-negative matrix with spectral radius r and row sums $r_1,\ldots, r_n$, then

$$\min_i r_i \leq r \leq \max_i r_i.$$

Hence, using Theorem 3.3, one choice for $\bar{t}$ is to use

$$\alpha_i = \max_i (Q - Qe_j e_j^T)_i e$$

which yields

$$\bar{t} = \frac{\ln(1-\delta)}{\ln(1-\min_{i,j} Q_{i,j})}.$$

Hence, if we can find the minimum element of the state transition matrix, Q, we will be able to find an upper bound on the number of iterations required for the first passage time probabilities. We will show that a unique minimum exists for matrix Q by proving two lemmas.

Lemma 5.1: The minimum element of matrix Q is bounded below by $\mu^{n\gamma}$.

Proof: We will prove Lemma 5.1 by proving two claims.

Claim 1: The minimum element of a multinomial distribution, where the minimum is being taken over all possible combinations of n_i values and where the multinomial

distribution is given by

$$P\{n_1,\ldots,n_k\} = n!\prod_{i=1}^{k}\frac{p_i^{n_i}}{n_i!}, \tag{10}$$

is

$$(\min_i\{p_i\})^n.$$

Proof of Claim 1: Suppose the p_i values are ranked in ascending order (i.e., p_1 is minimum). Then Claim 2 can be written as

$$\min_i P\{n_1,\ldots,n_k\} = P\{n,0,\ldots,0\}.$$

We will prove this by induction on n for a given k value.

Base Case: For n=1 Equation (10) can be written as

$$P\{n_1,\ldots,n_k\} = 1!\prod_{i=1}^{k}\frac{p_i^{n_i}}{n_i!}.$$

then

$$\min_i P\{n_1,\ldots,n_k\} = \min_i\prod_{i=1}^{k} p_i^{n_i}.$$

Finally by our assumption it follows that

$$\min_i P\{n_1,\ldots,n_k\} = P\{1,0,\ldots,0\} = p_1.$$

Inductive Hypothesis: Assume the claim is true for some arbitrary $n > 1$.

Now we need to show that it holds for n+1. That is

$$\min_i P\{n_1,\ldots,n_k\} = P\{n+1,0,\ldots,0\} = (p_1)^{n+1}.$$

By the induction hypothesis we know that

$$\min_i P\{n_1,\ldots,n_i,\ldots n_k\} = n!\frac{p_1{}^n p_2{}^0\ldots p_k{}^0}{n!\,0!\ldots 0!}.$$

Then

$$\frac{n+1}{n+1}p_1\min_i P\{n_1,\ldots,n_i,\ldots,n_k\} = \frac{n!(n+1)}{(n+1)}\frac{p_1 p_1{}^n p_2{}^0\ldots p_k{}^0}{n!\,0!\ldots 0!} \le \frac{n+1}{n+1}p_i P\{n_1,\ldots,n_k\}$$

$$\min_i \frac{n+1}{n+1}p_1 P\{n_1,\ldots,n_i,\ldots,n_k\} = \frac{(n+1)!}{(n+1)!}p_1{}^{(n+1)} = p_1{}^{n+1}.$$

Now consider the $Q_{i,j}$ expression defined earlier.

$$Q_{i,j} = n!\prod_{y=0}^{N-1}\frac{\left(\frac{Fz_i}{|Fz_i|}\right)^{Y_{y,j}}}{z_{y,j}!}.$$

Expanding the above equation gives

$$Q_{i,j} = n!\prod_{h=0}^{N-1}\frac{\left(\left(\sum_{i=0}^{N-1} f_i z_{i,h}\right)^{-2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_i z_{i,h} f_j z_{j,h} r_{i,j}(y)\right)^{Y_{j,h}}}{z_{j,h}}.$$

Consider

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_i z_{i,h} f_j z_{j,h} r_{i,j}(y).$$

Then

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_i z_{i,h} f_j z_{j,h} \min_{i,j} r_{i,j}(y) \le \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_i z_{i,h} f_j z_{j,h} r_{i,j}(y).$$

By using Lemma 2.2 we get

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_i z_{i,h} f_j z_{j,h} \min_{i,j} r_{i\oplus y,j\oplus y}(0) \le \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_i z_{i,h} f_j z_{j,h} r_{i,j}(y),$$

and, finally,

$$\min_{i,j} r_{i,j}(0) \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_{i\oplus y} z_{i\oplus y,h} f_{j\oplus y} z_{j\oplus y,h} \le \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_i z_{i,h} f_j z_{j,h} r_{i,j}(y).$$

Since the double sum on the left is constant, this motivates the need to determine the minimal element of M which consists of the $r_{i,j}(0)$ values.

Claim 3: The minimum element of Matrix M is bounded below by μ^γ, for $\mu < 0.5$.

Proof of Claim 3: Consider the expression for $r_{i,j}(0)$ representing the probability of producing 0 from parents i and j. This is given by

$$r_{i,j}(0) = \frac{(1-\mu)^\gamma}{2}\left[\eta^{|i|}\left(1-\chi+\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{-\Delta_{i,j,k}}\right)+\eta^{|j|}\left(1-\chi+\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{\Delta_{i,j,k}}\right)\right].$$

Rewriting the above expression we get

$$r_{i,j}(0) = \frac{(1-\mu)^\gamma}{2}\left[\eta^{|i|}(1-\chi)+\eta^{|j|}(1-\chi)+\chi\left(\frac{1}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{|i|-\Delta_{i,j,k}}+\frac{1}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{|j|+\Delta_{i,j,k}}\right)\right].$$

Consider the following inequalities for $\eta < 1$:

$$\eta^{|i|}(1-\chi) \geq \eta^{\gamma}(1-\chi) \text{ since } |i| \leq \gamma, \tag{11}$$

$$\eta^{|j|}(1-\chi) \geq \eta^{\gamma}(1-\chi) \text{ since } |j| \leq \gamma, \tag{12}$$

$$\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{|i|-\Delta_{i,j,k}} \geq \chi\eta^{\gamma} \text{ since } |i|-\Delta_{i,j,k} \leq \gamma, \tag{13}$$

$$\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{|j|+\Delta_{i,j,k}} \geq \chi\eta^{\gamma} \text{ since } |j|+\Delta_{i,j,k} \leq \gamma. \tag{14}$$

Note that the left hand sides of the first two inequalities represent the parents and the last two represent the children. Combining (11), (12), (13) and (14) we get

$$r_{i,j}(0) \geq \frac{(1-\mu)^{\gamma}}{2}\left[(1-\chi)\eta^{\gamma}+(1-\chi)\eta^{\gamma}+\chi\eta^{\gamma}+\chi\eta^{\gamma}\right]. \tag{15}$$

Finally simplifying (15) we get

$$r_{i,j}(0) \geq \frac{(1-\mu)^{\gamma}}{2}2\eta^{\gamma}.$$

Hence, since $\eta = \frac{\mu}{1-\mu}$,

$$r_{i,j}(0) \geq \mu^{\gamma}.$$

This proves Claim 3.

By combining Claim 2 and Claim 3 we obtain

$$Q_{i,j} \geq \mu^{n\gamma},$$

where n is as defined in Claim 1, and this proves Lemma 5.1. Q.E.D.

We will now prove that the bound is achievable, that is the minimum of Q is actually $\mu^{n\gamma}$.

Lemma 5.2: The minimum element of Q is $\mu^{n\gamma}$, for $\mu < 0.5$.

Proof: Consider the probability of producing the string 0 from parents N-1 and N-1. By applying crossover, the only possible child we can get is again N-1. Hence the only way we can get a 0 is to mutate all the bits. Then, the probability of getting a 0 from these two parents, is μ^{γ}, Thus $r_{N-1,N-1}(0) = \mu^{\gamma}$. So, the minimum of M is actually μ^{γ}. Hence, by using Claim 2, the lower bound given in Lemma 5.1. is tight. Q.E.D.

Finally we can derive an upper bound on the minimum number of iterations required for a desired level of confidence.

Corollary 5.1: The sufficient number of iterations required to find an optimal solution with a desired level of confidence δ is bounded above by

$$\frac{\ln(1-\delta)}{\ln(1-\mu^{\gamma n})}.$$

Proof: Combine Theorem 5.2 , Lemma 5.1 and Lemma 5.2 to get the desired result. Q.E.D.

5.3 Extensions and some Properties of the Bound

So far, while deriving the bound we have restricted ourselves to $\mu < 0.5$. It is possible to run the algorithm with very high mutation rates. So what happens if $\mu > 0.5$? We will show that increasing the mutation rate above 0.5 will not affect our results and we will derive the most general bound with a little modification.

Lemma 5.3: The minimum element of matrix Q is bounded below by $(1-\mu)^{n\gamma}$, for $\mu \geq 0.5$.

Proof: We will first prove a claim, Claim 4, and then combine it with Claim 2 to achieve the desired result.

Claim 4: The minimum element of matrix M is bounded below by $(1-\mu)^{\gamma}$, for $\mu \geq 0.5$.

Proof of Claim 4: Consider the expression for $r_{i,j}(0)$ representing the probability of producing 0 from parents i and j. This is given by

$$r_{i,j}(0) = \frac{(1-\mu)^{\gamma}}{2}\left[\eta^{|i|}(1-\chi+\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{-\Delta_{i,j,k}})+\eta^{|j|}(1-\chi+\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{\Delta_{i,j,k}})\right].$$

Rewriting the above expression we get

$$r_{i,j}(0) = \frac{(1-\mu)^{\gamma}}{2}\left[\eta^{|i|}(1-\chi)+\eta^{|j|}(1-\chi)+\chi(\frac{1}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{|i|-\Delta_{i,j,k}}+\frac{1}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{|j|+\Delta_{i,j,k}})\right].$$

Consider the following inequalities for $\eta \geq 1$.

$$\eta^{|i|}(1-\chi) \geq \eta^{0}(1-\chi) \text{ since } |i| \geq 0, \tag{16}$$

$$\eta^{|j|}(1-\chi) \geq \eta^{0}(1-\chi) \text{ since } |j| \geq 0, \tag{17}$$

$$\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{|i|-\Delta_{i,j,k}} \geq \chi\eta^{0} \text{ since } |i|-\Delta_{i,j,k} \geq 0, \tag{18}$$

$$\frac{\chi}{\gamma-1}\sum_{k=1}^{\gamma-1}\eta^{|j|+\Delta_{i,j,k}} \geq \chi\eta^{0} \text{ since } |j|+\Delta_{i,j,k} \geq 0. \tag{19}$$

Note that the left hand sides of the first two inequalities represent the parents and the last

two represent the children. Combining (16) ,(17) , (18) and (19) we get

$$r_{i,j}(0) \geq \frac{(1-\mu)^{\gamma}}{2}\left[(1-\chi)\eta^{0}+(1-\chi)\eta^{0}+\chi\eta^{0}+\chi\eta^{0}\right]. \qquad (20)$$

Finally simplifying (20) we get

$$r_{i,j}(0) \geq \frac{(1-\mu)^{\gamma}}{2} 2\eta^{0}.$$

Hence, since $\eta^{0} = 1$,

$$r_{i,j}(0) \geq (1-\mu)^{\gamma}.$$

This proves Claim 4.

By combining Claim 2 and Claim 4 we get ,

$$Q_{i,j} \geq (1-\mu)^{n\gamma},$$

where n is as defined in Claim 1, and this proves Lemma 5.3. Q.E.D.

We will now prove that the bound is achievable, that is the minimum of Q is actually $(1-\mu)^{n\gamma}$ for $\mu \geq 0.5$.

Lemma 5.4: The minimum element of Q is $(1-\mu)^{n\gamma}$, for $\mu \geq 0.5$.

Proof: Consider the probability of producing the string 0 from parents 0 and 0. By applying crossover, the only possible child we can get is again 0. Hence the only way we can get a 0 is not to mutate any of the bits. Then, the probability of getting a 0 from these two parents, is $(1-\mu)^{\gamma}$, Thus $r_{0,0}(0) = (1-\mu)^{\gamma}$. So, the minimum of M is actually $(1-\mu)^{\gamma}$. Hence, by using Claim 2, the lower bound given in Lemma 5.3. is tight. Q.E.D.

Finally we can derive an upper bound on the minimum number of iterations required for a desired level of confidence.

Corollary 5.2: The sufficient number of iterations required to find an optimal solution with a desired level of confidence δ is bounded above by

$$\frac{\ln(1-\delta)}{\ln(1-\min((1-\mu)^{\gamma n},\mu^{\gamma n})}.$$

Proof: Combine Theorem 5.2 , Lemma 5.1, Lemma 5.2 and Lemma 5.3 to get the desired result. Q.E.D.

It is possible to find the optimal values of μ and n for a given value of γ such that the minimum number of iterations required to claim that with certain probability the best solution in hand is optimal. That is, we can find optimal values of μ and n such that the bound obtained in Corollary 5.2 is tightest. So we can formulate the problem as a minimization problem (Problem 2.)

(Problem2) $$\min\ t = \frac{\ln(1-\delta)}{\ln(1-\min((1-\mu)^{\gamma n},\mu^{\gamma n})}.$$

subject to

$$0 \le \mu \le 1,$$

$$1 \le n \le 2^{\gamma},\ n \text{ integer.}$$

Instead of solving this problem we will solve the equivalent

$$\min\ t = \frac{\ln(1-\delta)}{\ln(1-\mu^{\gamma n})}.$$

subject to

$$0 \le \mu \le 0.5,$$

$$1 \le n \le 2^{\gamma},\ n \text{ integer,}$$

since $\min(\mu^{n\gamma},(1-\mu)^{n\gamma})$ is symmetric around $\mu = 0.5$.

Before we find the optimal solution to Problem 2, we will state three theorems that we will use to find the optimal.

Theorem 5.4 (Zangwill 1969): A differentiable function h: $E^n \rightarrow E^1$ is pseudoconcave if

$$\nabla h(x)^t(y-x) \leq 0$$

implies

$$h(y) \leq h(x).$$

A function h is termed pseudoconvex if -h is pseudoconcave.

Definition 5.1 A function h: $E^n \rightarrow E^1$ is called quasi-concave if given $x^1, x^2, \in E^n$, for any Θ, $0 \leq \Theta \leq 1$,

$$h(\theta x^1 + (1-\theta)x^2) \geq \min[h(x^1), h(x^2)].$$

A function h is quasi-convex if -h is quasi-concave. Any pseudoconcave function is quasi-concave.

Theorem 5.5 (Bazaraa and Shetty 1979): Let S be a nonempty polyhedral set in E^n, and let f: $E^n \rightarrow E^1$ be quasi-convex and continuous on S. Consider the problem to maximize f(x) subject to $x \in S$. Then there exists an optimal solution x^* to the problem, where x^* is an extreme point of S.

Theorem 5.6: The optimal solution of Problem 2 is $x^* = (\mu^*, n^*) = (0.5, 1)$.

Proof: We will first show that t is pseudoconcave for fixed n. The partial derivative of t with respect to μ is given below:

$$\frac{\partial t(\mu,n)}{\partial \mu} = \frac{\ln(1-\delta)n\gamma\mu^{n\gamma-1}}{(1-\mu^{n\gamma})\ln^2(1-\mu^{n\gamma})}. \qquad (21)$$

It is clear that for feasible values of μ and for fixed n (21) is always negative. Select two feasible points $x,y \in \mathbb{R}^2$ such that $y = x + (\alpha,0)$ where α is positive. Clearly x and y will satisfy

$$\nabla h(x)^t(y-x) \leq 0.$$

It will suffice to show that $h(x) \geq h(y)$ holds.

We have to show that

$$\frac{\ln(1-\delta)}{\ln(1-\mu^{n\gamma})} \geq \frac{\ln(1-\delta)}{\ln[1-(\mu+\alpha)^{n\gamma}]}. \tag{22}$$

(22) is equivalent to

$$\mu^{n\gamma} \leq (\mu+\alpha)^{n\gamma}. \tag{23}$$

Assume (23) is not correct. Then

$$\ln(\mu+\alpha) \leq \ln(\mu),$$

which is a contradiction. Hence the claim is true. So t is pseudoconcave for fixed n. By definition t is quasi-concave and -t is quasi-convex for fixed n.

Instead of solving Problem 2 we will solve Problem 3 which is defined below. Fix n such that $n = n_0$, then

(Problem 3)

$$-(\max -t(\mu,n_0)$$

subject to

$$0 \leq \mu \leq 0.5,$$

$$n = n_0,\ n_0 > 0,\ n \text{ integer.}$$

Since $t(\mu,n_0)$ is quasi-concave $-t(\mu,n_0)$ is quasi-convex. The constraint set is a non-empty compact polyhedral set. Then, by Theorem 5.5 the solution to this problem will occurs at the extreme points. The set has two extreme points $(0,n_0)$ and $(0.5,n_0)$. Clearly the minimum occurs at $(0.5,n_0)$.

Now we have to show that $t(\mu^*,n) \leq t(\mu^*,n+1)$. It will suffice to show that

$$(\mu^*)^{n\gamma} \geq (\mu^*)^{(n+1)\gamma}. \tag{24}$$

Assume (24) is not correct. Then

$$n\gamma \geq (n+1)\gamma,$$

which is a contradiction. Then $t(\mu^*,n) \leq t(\mu^*,n+1)$. This proves that the minimum will occur at $n = 1$. Hence, the solution to Problem 3 is $t(\mu^*,n^*) = (0.5,1)$. Q.E.D.

CHAPTER 6
LEARNING DISPATCHING RULES FOR SCHEDULING USING SIMULATION

6.1 Overview

In previous chapters we tried to motivate the need for an intelligent Decision Support System (DSS), that is able to learn adaptively. In this chapter we show how machine learning methods can be integrated within a simulation environment to schedule. The hope is that this approach will relieve the complex search needed with a pure simulation model and enable one to avoid the need to acquire new knowledge using time-consuming AI knowledge-acquisition approaches. The system will teach itself how to run the environment.

Note that this approach bears some similarity to perturbation analysis, in that changes are made as the simulation progresses. However, no objective function is needed and no gradients are estimated. Furthermore, we permit a more flexible simulation environment where inference drives decisions - not rigid rules (such as queuing disciplines).

The main paradigm shift offered in our study is the explicit incorporation of intelligent objects in the simulation environment. These objects can draw both deductive and inductive inferences to improve the process as the simulation progresses. In addition, these objects can alter the very nature of the environment. For example, we permit the dispatcher to dynamically change the queue discipline.

To illustrate this adaptive learning process, a simple simulation environment is considered. Standard dispatching rules are selected using a knowledge-base. Based upon a performance measure supplied by a user, a learning device forms a new knowledge-base. The study showed that, for a given performance measure, scheduling jobs using a learning approach out-performed two baseline runs using first-come, first-served and earliest due date rules. Future research issues are discussed.

In Section 6.2 a conceptual model of the environment to be studied is presented. In Section 6.3, the learning algorithm is discussed. The simulation environment is discussed in Section 6.4. In Section 6.5 an experiment is described and results are presented. In Section 6.6 the results of this study are discussed. Several issues raised by these results are explored further in Section 6.7.

6.2. Conceptual Model

In this section, a system is described as a collection of objects that interact with one another. Some objects are identified as intelligent objects. Intelligent objects make decisions that affect the system's performance. Other objects (like queues, jobs, servers, etc.) react as they are designed and as chance dictates.

Figure 6-1 shows an example that illustrates this approach. It is a simple setting consisting of 5 parallel machines with different rates. Jobs (job objects) enter a queue (QUEUE 1 - a queue object). Each job is stamped with its entry time and due date. Due dates are generated randomly using a uniform distribution.

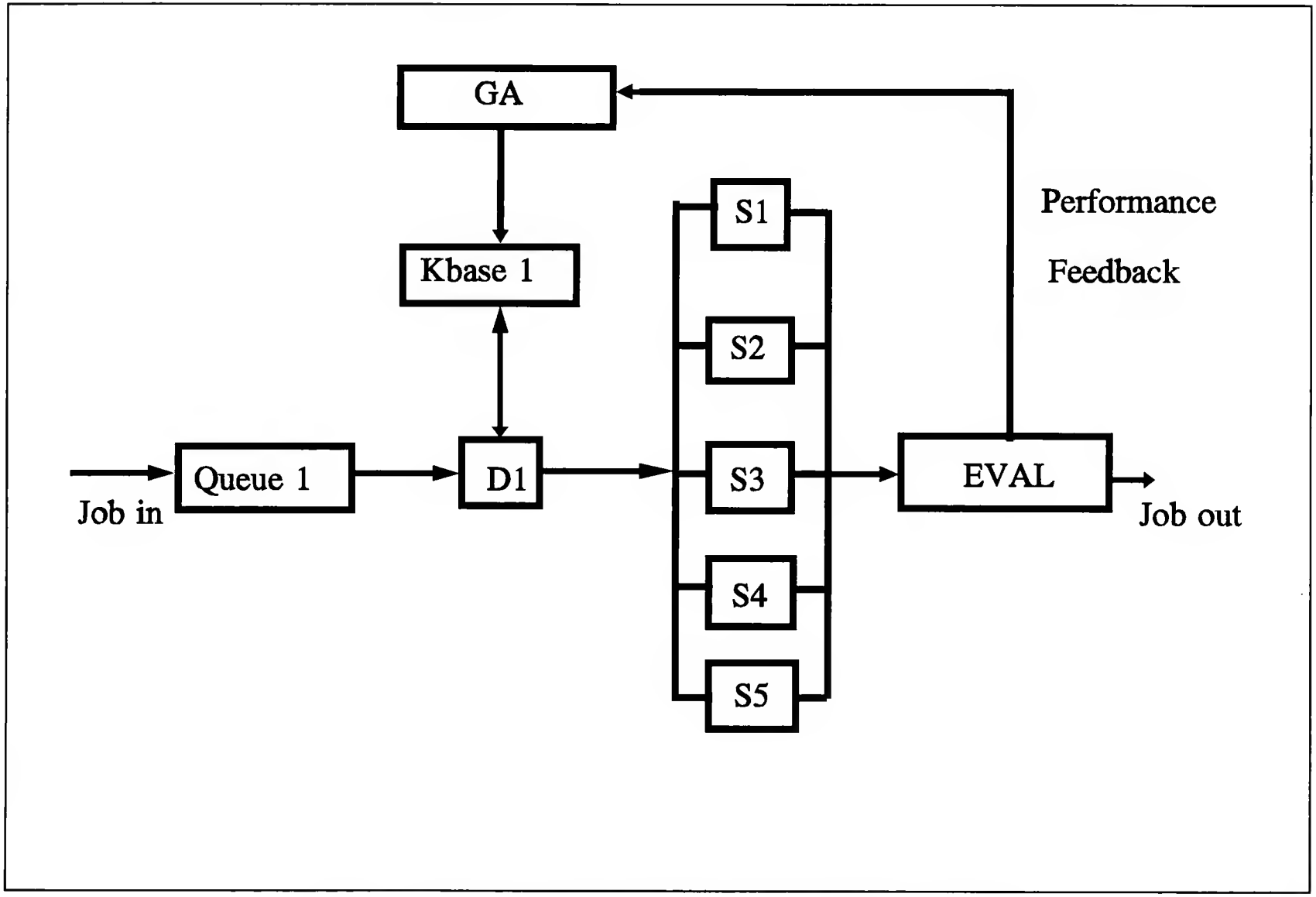

Figure 6-1 GA Based Learning System in a Simulation Environment

Whenever a server object (servers are marked S1, S2, S3, S4, S5) is free, a dispatcher object (D1) decides which job in the queue will be released for service. The dispatcher is an example of an intelligent object.

Intelligence is exhibited by decision making. Decision making may be the result of algorithmic processes, table lookups, inference or other processes (such as flipping a coin).

When a system is being studied by simulation methods, the decision-making objects are the focus of the researcher's attention. The researcher must decide how decisions are to be made to optimize the system's performance.

In the absence of theoretical guidance or computing time limitations (for NP hard problems), deciding how decisions are to be made to optimize a system's performance is a time-consuming task, often involving large numbers of carefully designed experiments.

Here is presented a machine learning approach that will enable the intelligent objects to learn how to make better decisions. Intelligent objects will be endowed with a knowledge-base and learning component. In Figure 6-1, the dispatcher object owns a knowledge base (Kbase 1) and a learning element (called GA).

When a server becomes idle and the input queue is not empty, the dispatcher uses an inference process with his knowledge base to decide which job to route to the idle server.

Servers of the same variety are assumed to be able to perform the same activities but may do so with different rates. Also, their breakdown and repair rates may be

different. In general, servers can be endowed with a host of different attributes. The servers in Figure 6-1 are assumed to be of the same variety but have different service, repair and breakdown rates.

When a server completes a job, the job moves to an evaluation element (EVAL). The evaluation component can be a simple cost model, a model measuring the systems performance based on various performance elements, or an inference against a knowledge base that captures management's complex way of trading off multiple, and sometimes conflicting, objectives. The evaluation component provides a feedback mechanism to the learning component. Periodically, the learning component updates the knowledge base to reflect the results of past decisions and their impact on the system's performance.

6.3. Genetic Learning and Inference

The learning component is a classifier system based on a Genetic Algorithm (GA). In particular, it employs the Michigan approach (Holland, 1986). Below are described the specifics of the rule structure, inference process and learning method.

An initial knowledge base is either randomly generated or provided by the usual knowledge-acquisition methods. As decisions are made, the jobs are tagged by the knowledge used in their selection. After the jobs are processed and the system's performance evaluated for this job, the responsible knowledge is credited with performance (which may be good or poor).

Periodically, the cumulative performance of each piece of knowledge is used to form a new knowledge base using reproduction, crossover and mutation operators by the

GA. Through this "survival-of-the-fittest" approach, we show that the successive generations of knowledge tend to improve their performance.

A rule is defined using conjunctions of NDTV + 1 truth values. Here, NDTV stands for the number of dispatching rule truth values (described below). The first truth value is of the form:

(Server_ID == i)

which evaluates to TRUE or FALSE. Here == means "equal to." Each rule is associated with a particular server. The remaining NDTV truth values have the following structure:

JOB == d_rule()

or

(*),

where d_rule() is a standard heuristic dispatching rule. It can be thought of as a function that returns a job id that satisfies its requirements. When multiple jobs satisfy the criterion, ties are broken by a First-Come, First-Served decision. If no job satisfies the criterion, a NULL value is returned. The arguments for the function are suppressed but may include objects from the environment. Thus, if a specified JOB is identical to the job selected by the dispatch rule, the expression evaluates to TRUE.

(*) is TRUE for any JOB.

There are many heuristic dispatching rules (Blackstone, Phillips and Hogg, 1982). The dispatching rules included in this study are:

FCFS - First-Come, First-Served.

SI - Shortest Imminent operation.

LI - Longest Imminent operation.

FRO - Fewest Remaining Operations.

MRO - Most Remaining Operations.

SR - Shortest Remaining processing time.

LR - Longest Remaining processing time.

TO/I - minimum Total Operations divided by Imminent duration.

RPT/I - minimum Remaining Processing Time divided by Imminent time.

GTW - Greatest Total Work.

EDD - Earliest Due Date.

MST - Minimum Slack Time.

SLACK/OPN - Least SLACK per OPeratioN.

JSR - Job Slack Ratio.

Since there was only one operation in the simple model (Figure 6-1), the following are equivalent to FCFS in our environment:

SI, LI, FRO, MRO, SR, LR, RPT/I and GTW.

For NDTV set to 2, a few examples of rules are:

(Server_ID == S2) ^ (*) ^ (JOB == EDD()).

This rule evaluates to TRUE if server S2 is IDLE and JOB is the job with the earliest due date.

(Server_ID == S1) ^ (*) ^ (*).

This rule evaluates to TRUE for any JOB if the idle server is S1.

(Server_ID == S3) ^ (JOB == MST()) ^ (JOB == EDD())

This rule evaluates to TRUE if server S3 is IDLE and JOB is the job with the earliest due date AND the minimum slack time.

A server is randomly picked from the list of idle servers. If a rule evaluates to TRUE for this server, it may vote for the job it selected. This inference process relies on bidding. All rules cooperate on the selection of each job. Other examples of bidding systems can be found in (Shaw, 1986), (Shaw, 1988).

After all the votes are tallied, a job is picked randomly according to the voting frequencies. (This inference process allows for cases where an optimal strategy may be a mixed strategy.) If no job receives a vote, a JOB is selected randomly.

The selected job is then tagged with all rules that voted for it. These rules will receive feedback as the job exits the system. The feedback is reflected in the rule's performance from which fitness measures are computed.

Ideally, the performance measure will reflect management's evaluation of and trade-offs between a number of criteria. In this study, no "management" exists to supply evaluation criteria. In the absence of such criteria, the system's performance on each job is measured by a service quality criteria. Before we describe the performance function we will introduce some notation. Let,

$p_{i,j}$ be the performance of rule i on job j;

p_j be the attained performance on job j.

$\bar{p}_i$ be the average fitness of rule i. That is, $\bar{p}_i = \sum_{j \in J_i} p_{i,j}$, where J_i is the set of jobs for which rule i voted in between to learning episodes;

f_i be the fitness of rule i.

r_j be the time release time of job j;

ts_j be the time the service starts on job j;

te_j be the time when service ends on job j and it leaves the system;

and

t be the current time.

The performance of the knowledge base on any job is given by

$$p_j = \frac{te_j - ts_j}{te_j - r_j}$$

A rule's total performance is an average over the performance for each job tagged by the rule. Its fitness is computed by

$$f_i = \bar{p}_i^{\,3}$$

This measure of fitness severely penalizes poor performance. The exponent is called the emphasis coefficient. The fitness measure is used by GA to randomly select rules from the current knowledge base. The higher a rule's fitness, the more likely it will be chosen for mating or survival.

During knowledge base updating, a rule is selected randomly based on the fitness measures. A second rule is then selected randomly, again based on fitness, but with the restriction that it has the same Server_ID.

Once selected, the two rules either mate (with the "crossover" probability) or survive. Mating involves randomly picking a crossover point (i.e., randomly picking a truth value term at conjunction position 1,...,NDTV) and swapping the truth value terms of the two rules starting at this point. For example, the two mating parents

(Server_ID == S3) ^ (JOB == MST()) ^ (JOB == EDD())

(Server_ID == S3) ^ (JOB == FCFS()) ^ (JOB == SI())

might crossover at the second truth value term (after the Server_ID term) yielding the children:

(Server_ID == S3) ^ (JOB == MST()) ^ (JOB == SI())

(Server_ID == S3) ^ (JOB == FCFS()) ^ (JOB == EDD())

Surviving parents or newly created children rules are then subject to mutation (with a "mutation" probability). If selected for mutation, a randomly picked truth value term at position 1,...,NDTV is randomly altered to another term. For example, if the rule

(Server_ID == S3) ^ (JOB == MST()) ^ (JOB == SI())

were selected for mutation and mutation were to occur at the first truth value term (after the Server_ID term), the following might result:

(Server_ID == S3) ^ (JOB == EDD()) ^ (JOB == SI())

This process continues until a new knowledge base of the same size is created. Restrictions were added so that there was an equal number of rules in the knowledge base for each Server_ID. This did not impact the solution.

The crossover and mutation operators are similar in idea to the "integer approach" of Frey and Slate (1991). Both operators are restricted to the dispatch rule fragments. That is, only rules with the same server may mate. However, an encoding scheme on the alphabet {0,1} could have been employed as follows. Assume there are m servers, d dispatching rule, and the number of conjuncts is NDTV. Then total number of possible combinations of rules is given by

$$\sum_{i=0}^{NDTV} \binom{d}{i}, \quad NDTV \leq d.$$

and given that we distinguish between servers total number of rules we need to search is

$$m \sum_{i=0}^{NDTV} \binom{d}{i}.$$

Finally by using a string of length

$$\log_2 \left(m \sum_{i=0}^{NDTV} \binom{d}{i} \right)$$

we can represent all possible strategies. For implementational convenience we employed the integer coding.

If a rule is to be mutated, a truth value component is randomly chosen and then replaced with a randomly chosen (different) dispatch rule (which may be the wild-card, *). Two population members are mated by randomly choosing a crossover point and then switching the trailing fragments. In this study, the mutation rate is 0.005 and the crossover rate is 0.6. These values are typical values found useful in a number of empirical GA studies. Since our aim is simply to try to demonstrate the applicability of GA based learning, we did not perform a sensitivity analysis on these parameters. As will be clear later, the solutions obtained were quite satisfactory.

6.4. Simulation Environment

The simulation environment is an event-oriented model and was implemented with object-oriented programming (OOP) using C++. The following objects were defined:

CLASS	Description
Node	Node objects are used to link other objects.
List	A list is a string of linked objects.

Queue	A queue object is a list following some ordering (first in, first out - FIFO, last in, last out - LIFO, or SORTED on an attribute) that may be attached to servers, other queues and a dispatcher.
Server	A server object operates on jobs.
Rule	A rule object encodes a rule in the knowledge base language.
Kbase	A kbase object is a collection of rules.
Dispatcher	A dispatcher object decides which job will be selected and processed by an idle server.
Event	An event object is placed on the future event list (fel) and indicates what is to be done and when.
Job	A job object moves through the simulation environment.

Figure 6-2 shows the relationships among the various classes of objects. (In C++, a FRIEND object has access to another object's private methods and data fields.)

The simulation environment is structured as follows. First, all server objects, queues, dispatchers, and the future events list (fel) are declared. Next the fel is initialized with:

. the first job arrival;

. the first breakdown for each server;

. the simulation termination;

and

. the time of the first update of the knowledge base.

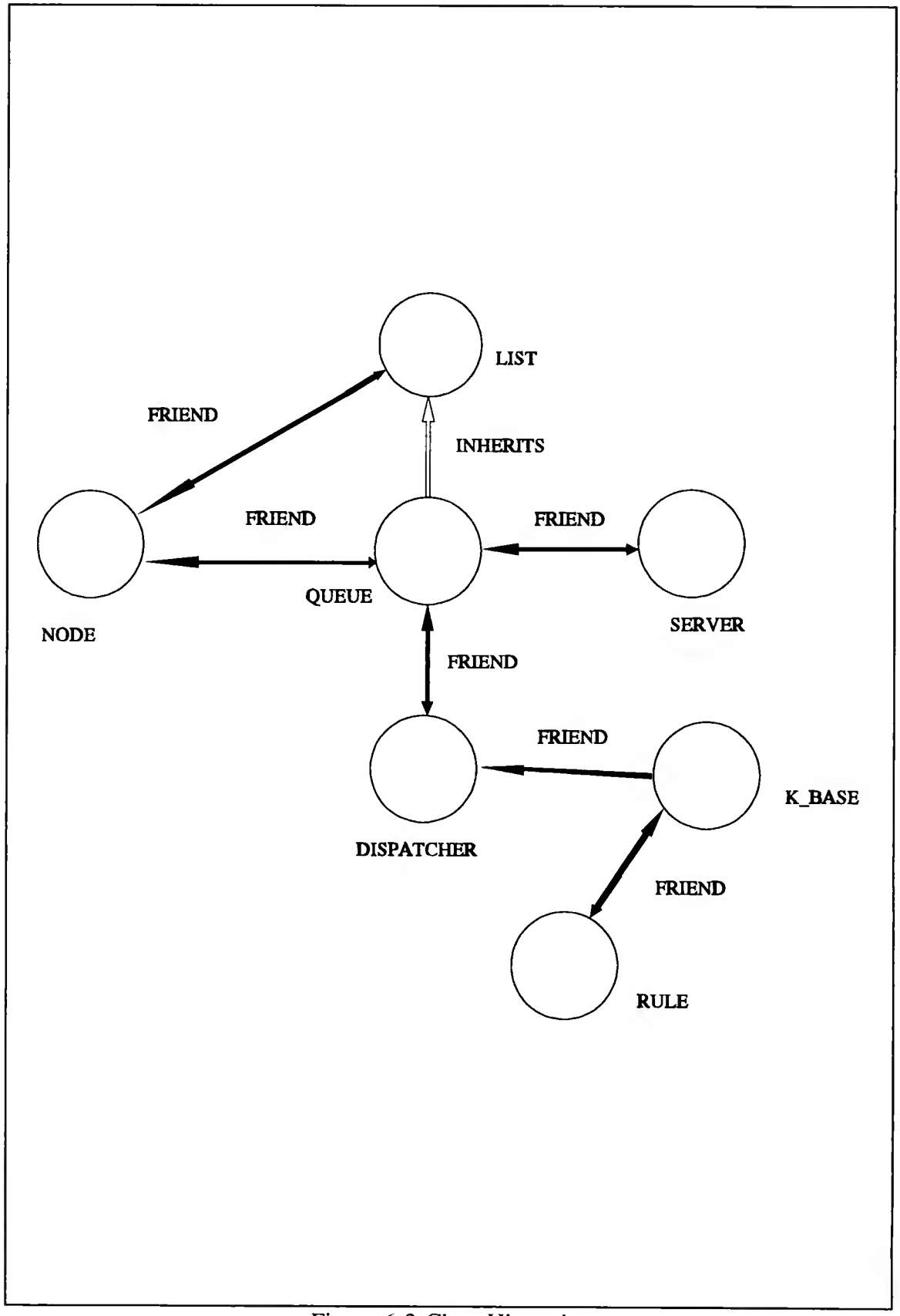

Figure 6-2 Class Hierarchy

Finally, the topology of the simulation environment is defined.

The following activities are performed in a simple event-oriented loop as the simulation executes.

EVENT	Description
ARRIVAL	A new job is created and a next arrival generated. Jobs are created with various attributes (priority, due date, etc.)
SETUP_START	If there is an IDLE server, then the dispatcher is asked to select a queue member. This member is removed, linked to the server and a SETUP_END is scheduled if no server breakdown will occur during setup.
SETUP_END	Relevant statistics are updated and a SERVER_START event is scheduled.
SERVER_START	A SERVER_END event is scheduled if no breakdown will occur during service. Statistics are updated.
SERVER_END	Relevant statistics are updated. The job is transferred from the in-service buffer to the served-jobs buffer. If the preceding queue still has members, a SETUP_START is scheduled. An ARRIVAL to the next queue is scheduled if one exists. Otherwise a LEAVE event is scheduled.

BREAK_DOWN_START	Relevant statistics are updated. If a server is IDLE when a breakdown occurs, then the server is marked as BUSY and an BREAK_DOWN_END is scheduled. Otherwise, and additionally, a flag is set to show that it failed with a job in process.
BREAK_DOWN_END	Relevant statistics are updated. If a job was being processed at the failure time, a SERVER_START is scheduled. Otherwise, a SETUP_START is scheduled and the server is marked IDLE.
LEARN	The genetic algorithm is used to form a new knowledge base. The next LEARN is scheduled.
LEAVE	A job leaves the system. Various statistics are captured. If the current knowledge base was responsible for routing this job, the performance on this job is used in credit assignment.
TERMINATE	Statistics are computed and printed. The final knowledge base is printed. The program is terminated.

The initial knowledge base is generated as follows. First, NDTV is specified. Then, for each server, rules are built by generating all combinations of rules having, at most, one dispatching rule.

For example, suppose NDTV is set to 2 and only FCFS, EDD and SI dispatching rules were used. The initial knowledge base for server S1 would be:

(Server_ID == S1) ^ (*) ^ (*)

(Server_ID == S1) ^ (JOB == FCFS()) ^ (*)

(Server_ID == S1) ^ (JOB == EDD()) ^ (*)

(Server_ID == S1) ^ (JOB == SI()) ^ (*)

(Server_ID == S1) ^ (*) ^ (JOB == FCFS())

(Server_ID == S1) ^ (*) ^ (JOB == EDD())

(Server_ID == S1) ^ (*) ^ (JOB == SI()).

The initial rule base is generated this way so that each dispatching rule is potentially available in a crossover. Otherwise, a given dispatching rule at a specified truth value position could only be created through mutation. The sequence of the dispatching rules is immaterial since crossover is a random operator.

There is no clear-cut way to select NDTV. NDTV is bounded from above by the number of dispatching rules. In the extreme, this would permit a rule composed of the conjunction of all the dispatching rules. Two lines of thought argue against large values of NDTV. First, the "Occam Razor" principle of scientific research argues for as small a value of NDTV that gives reasonable results. Second, larger values of NDTV will often yield rules that are inconsistent. In effect, we would have to search a larger space having a larger percentage of inconsistent rules.

A value of NDTV equal to one would have given rules equivalent to mere dispatching rules. In our study we chose NDTV equal to two. This permitted more involved rules than mere dispatching rules and provided good results. Unreported results using NDTV equal to three were not significantly better than with the value of two.

6.5. Experiment and Results

To illustrate the idea presented here, Figure 6-1 was declared and three simulations were run for 3,000 time units each. Running the system longer produced no discernible change in performance. The three runs include a GA learning run plus two baseline runs (FCFS dispatching and EDD dispatching only). A new knowledge base was generated every 40 time units. In 40 time units, an appreciable number of jobs were dispatched and serviced, resulting in reasonable estimates of the value of individual rules.The simulation started at time zero. There was no initial "warming-up" period.

The specific scenario implemented in Figure 6-1 is described as follows. The dispatcher knows an existing job's arrival time and due date, the list of idle servers and the expected service time of each server. Different servers may have different speeds but the expected service time for each job is the same for a given server (which accounts for the equivalence between LI and SI and between LR and SR dispatching rules). Neither future arrival times or actual service times are known by the dispatcher. Both arrival and service processes are independent Poisson.

All arrivals, breakdowns, repairs and service times are exponential random variables. Server values are:

Server	SERVICE	MTTF	MTTR
S1	17.333	480.7	1.6
S2	17.400	1284.2	2.0
S3	30.333	849.2	2.5
S4	24.000	447.8	0.3
S5	28.333	945.6	0.3

and the arrival rate was 100 per time unit. NDTV was set to 2 in this case, but is not limited to two in general.

In Figure 6-3 the population's average performance (Y-axis) is shown over time (X-axis).The population average performance was computed using all those jobs dispatched and serviced during the 40 minute learning epoch.

It is easy to see that after a short period, the system is performing at about an 82% level under GA learning and that GA learning dominates the two baselines of FCFS and EDD. In turn, EDD dispatching dominates FCFS dispatching. The final knowledge base contained (sometimes multiple copies of) the following rules:

(Server_ID == S1) ^ (JOB == JSR()) ^ (JOB == TO/I())

(Server_ID == S2) ^ (JOB == FCFS()) ^ (JOB == EDD())

(Server_ID == S3) ^ (JOB == FCFS()) ^ (JOB == TO/I())

(Server_ID == S4) ^ (JOB == FCFS()) ^ (JOB == MST())

(Server_ID == S5) ^ (JOB == TO/I()) ^ (JOB == SLACK/OPN())

(Server_ID == S5) ^ (JOB == FCFS()) ^ (JOB == SLACK/OPN())

The first rule is interpreted as follows.

For server S1 assignments, if a JOB is both optimal with respect to JSR and TO/I, then it is chosen. Otherwise, a job is picked randomly from the queue. Do these rules persist?

A run was also performed that went for 20,000 time units. All the rules persisted with one exception for server S5. The second rule died off.

Can the learning component improve performance in a congested system?

To answer this question, the arrival rate was increased to 110 jobs per time unit. This resulted in a FCFS performance of about 41% - down from 63%. As can be seen in Figure 4, the system performed best with learning. EDD still dominated FCFS. While these dominations are clear, the system seems much more sensitive to rare events, like machine breakdowns, which caused the sharp downward spikes. Also, at several points (other than the initial few periods), EDD had a better performance than did the system using learning. The final knowledge base contained (sometimes multiple copies of) the following rules:

(Server_ID == S1) ^ (JOB == FCFS()) ^ (JOB == JSR())

(Server_ID == S2) ^ (JOB == FCFS()) ^ (JOB == SLACK/OPN())

(Server_ID == S3) ^ (JOB == FCFS()) ^ (JOB == TO/I())

(Server_ID == S4) ^ (JOB == FCFS()) ^ (JOB == MST())

(Server_ID == S4) ^ (JOB == FCFS()) ^ (JOB == JSR())

(Server_ID == S4) ^ (JOB == JSR()) ^ (JOB == MST())

(Server_ID == S5) ^ (JOB == FCFS()) ^ (JOB == EDD()).

6.6. Discussion of Results

Figure 6-3 shows that the knowledge base evolved to give an average performance that dominated the two baseline runs using the FCFS and EDD rules, respectively. Learning produced dispatching that achieved an 82% performance (approximately). EDD led to 70% performance (approximately) while FCFS had a 63% performance (approximately).

With a more congested environment, Figure 6-4 shows that the knowledge base evolved to give an average performance that still dominated the two baseline runs using the FCFS and EDD rules, respectively. Learning produced dispatching that achieved a 68% performance (approximately). EDD led to 53% performance (approximately) while FCFS had a 41% performance (approximately).

The resulting knowledge base is rather interesting. Almost every rule has A FCFS conjunct and a due date related measure. It is known that FCFS is a good heuristic to minimize the variance of the average waiting time and in this study because of the way due dates were assigned, due date related rules had an inherent superiority over FCFS. We are unable to prove it analytically but the learning element seems to have discovered these characteristics and use due date related heuristics for mean performance and FCFS for reducing variability.

Refer to our claim above about EDD performing better than FCFS. One question that arises is: Since the performance measure is not a function of due date, why does EDD out-perform FCFS? The answer lies in the choice of performance measure.

Recall that

$$p_j = \frac{te_j - ts_j}{te_j - r_j}.$$

In this model, a given server will process any job presented to it at a given time with the same service time regardless of what dispatching rule is used. (Of course, the service time itself is a random variable). So, at a given dispatch decision, tss_j, the total time spent in service by job j, will be the same for any selected job.

Now, we can rewrite total time spent in system as the sum of time spent in queue, $ts_j - r_j$, and tss_j. Hence, the performance measure for job j selected by the dispatcher will be:

$$p_j = \frac{tss_j}{ts_j - r_j + tss_j}$$

FCFS will always select the job (subscripted by fcfs) with the lowest r_j value. EDD may select this job, but may select another with an earlier due date (with a later arrival time). Subscript this job using edd. Then

$$r_{fcfs} \leq r_{edd}.$$

In this environment

$$ts_{fcfs} = ts_{edd}$$

and

$$tss_{fcfs} = tss_{edd}.$$

But then

$$ts_{fcfs} - r_{fcfs} + tss_{fcfs} \geq ts_{edd} - r_{edd} + tss_{edd}.$$

The denominator of performance under FCFS dispatching will be greater than or equal to the denominator of performance under EDD. Thus,

$$p_{fcfs} \equiv \frac{tss_{fcfs}}{ts_{fcfs} - r_{fcfs} + tss_{fcfs}} \leq \frac{tss_{edd}}{ts_{edd} - r_{edd} + tss_{edd}} \equiv p_{edd}.$$

and EDD's performance will be better than the performance of FCFS.

Figure 6-4 has several points of interest worth discussing. First, while system performance decreases with congestion, the results under learning seem to decrease at a slower rate and hence improve relative to FCFS and EDD.

A second point of interest is that the system was very sensitive to rare events, like machine breakdowns. This is true for each type of control - whether for FCFS, EDD or learning. An obvious insight is to include factors into the rule structure that could anticipate these types of occurrences. For example, a truth value of the form

$$(\text{time\_since\_last\_breakdown} \geq \text{XXX})$$

could be included where XXX is subject to GA learning.

Another point of interest is that the performance of the system under learning "bounced-back" after the unexpected machine breakdown.

6.7. Related Issues

The results presented in this chapter appear very promising. However, several issues need to be resolved. Below we address the following topics:

1. Are the rules really selecting jobs or are they usually being chosen randomly?
2. The systems appears to adapt quickly to changes. Does this persist under other types or rates of changes?
3. Does the GA learning scheme discover optimal policies when such are known to exist?

To facilitate addressing these questions, some changes were made in the simulation environment shown in Figure 6-1. The first was to make all servers identical. This both

enabled us to study question (3) and to remove possible obfuscation due to major differences in the servers. At the start of a simulation run, an expected service rate was set by drawing a uniform random number from [0.03, 0.05]. The actual service times were exponential with this value.

The second change was to increase the failure rate which helped in studying Question (2). In Figure 6-5 we show the performance of the system under increasingly frequent machine breakdowns. Here MTTF/# means divide the MTTF values used earlier by #. Notice that the performance actually increases with more frequent breakdowns. Below we report all runs with MTTF/3 rates. Figure 6-6 shows the performance of the learning system compared to the FCFS and EDD baselines. This is a repeat of Figure 6-3 under the new conditions.

Question 6.1

Two runs were made to address the first question: Are the rules really selecting jobs or are they usually being chosen randomly? In the first run, voting behavior for the results displayed in Figure 6-3 was observed. There were a high number of times where the inference process relied on random choice, but the system "settled-down" to random choices only when the rules didn't identify significant features. This led to asking whether a purely random policy would perform as well as GA learning.

In the second run, the GA learning method was compared to the dispatching rule: SIRO - Service In Random Order.

The results are shown in Figure 6-7. GA learning performs at about the 89% level where SIRO only performs at the 60% level.

We conclude that GA learning is taking place, that rules are being used and that GA learning is performing significantly better than mere random choices.

Question 6.2

The systems appears to adapt quickly to changes. Does this persist under other types or rates of changes? We have seen above that the system responded to increasing rates of machine breakdowns. We also studied other types of changes.

In Figure 6-8 we show the system running under non-homogeneous Poisson arrivals. Here inter-arrival times were computed using

Rate * (1 - .3 * cosine(2 * Time))

where RATE is the value used earlier. GA learning still dominates. The four runs give:

Run	Performance
GA Learning	78%
FCFS	55%
EDD	59%
SIRO	56%.

It appears that GA learning is very adaptive.

Question 6.3

The third question concerns optimality: Does the GA learning scheme discover optimal policies when such are known to exist?

To study this question, the system was altered as follows. For each learning cycle, a specified number of jobs was released into the system. At the time of their release they were tagged with a processing time that was randomly generated from a uniform distribution over [0.3, 0.9]. The performance measure involved a weighting of the GA

learning order versus the order required under the SI rule (which is known to minimize flow time in this case.) Note that this is a performance measure different from that used in the rest of this paper.

In a series of runs, the number of jobs (ranging from 10 to 120) released and the number of learning cycles (ranging from 100 to 200) was varied. All gave similar results. Figure 6-9 shows the results of a typical run. Here we show 120 jobs being released in each of 50 learning cycles (the first 50 of a 200 cycle run.) Notice that the GA learning quickly converges to the optimal policy. Figure 10 shows the same environment run with GA learning versus FCFS, EDD and SIRO baselines. GA learning out-performed each.

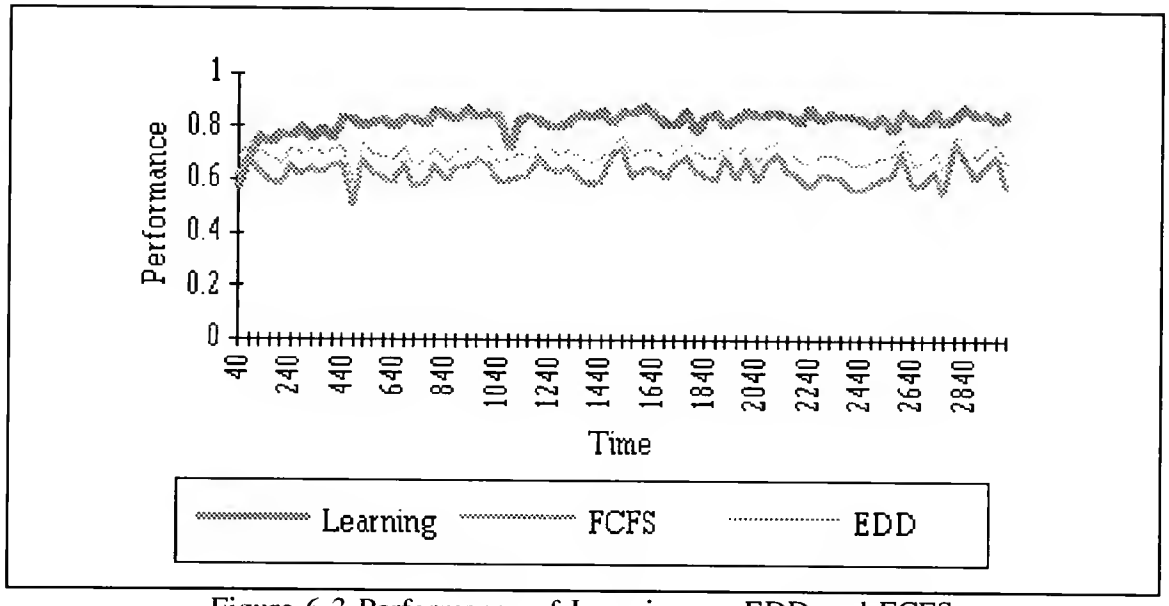

Figure 6-3 Performance of Learning vs. EDD and FCFS

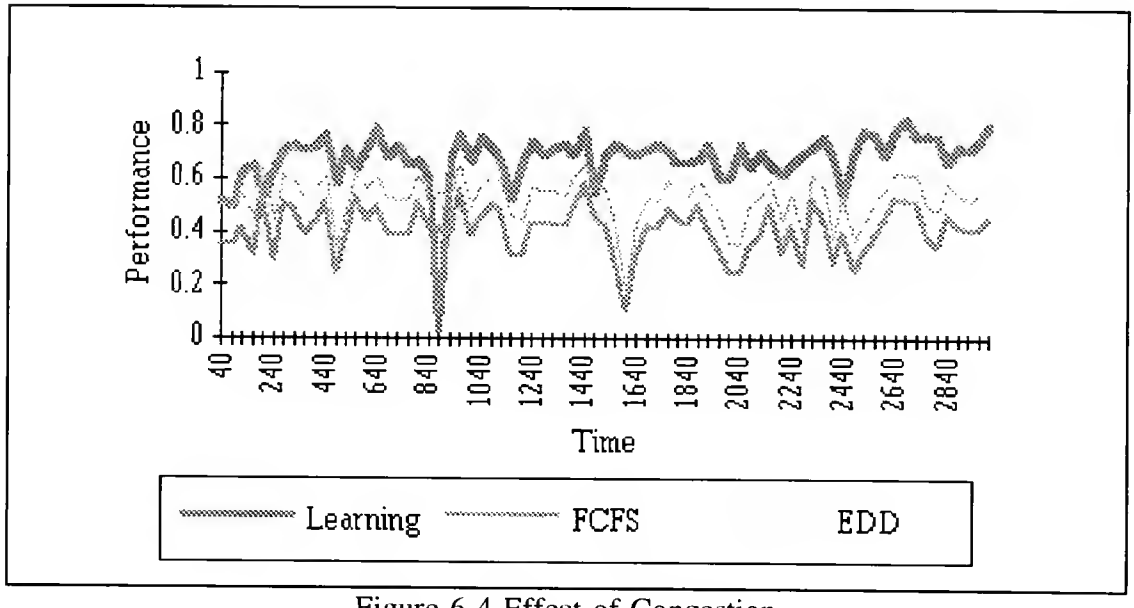

Figure 6-4 Effect of Congestion

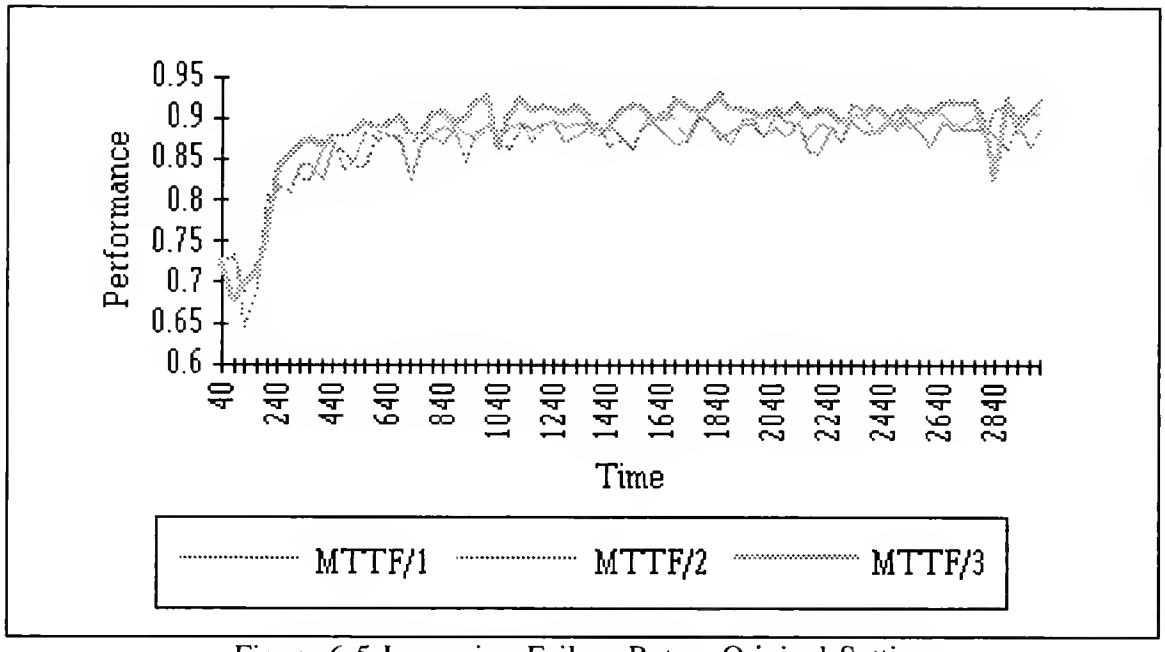

Figure 6-5 Increasing Failure Rates--Original Setting

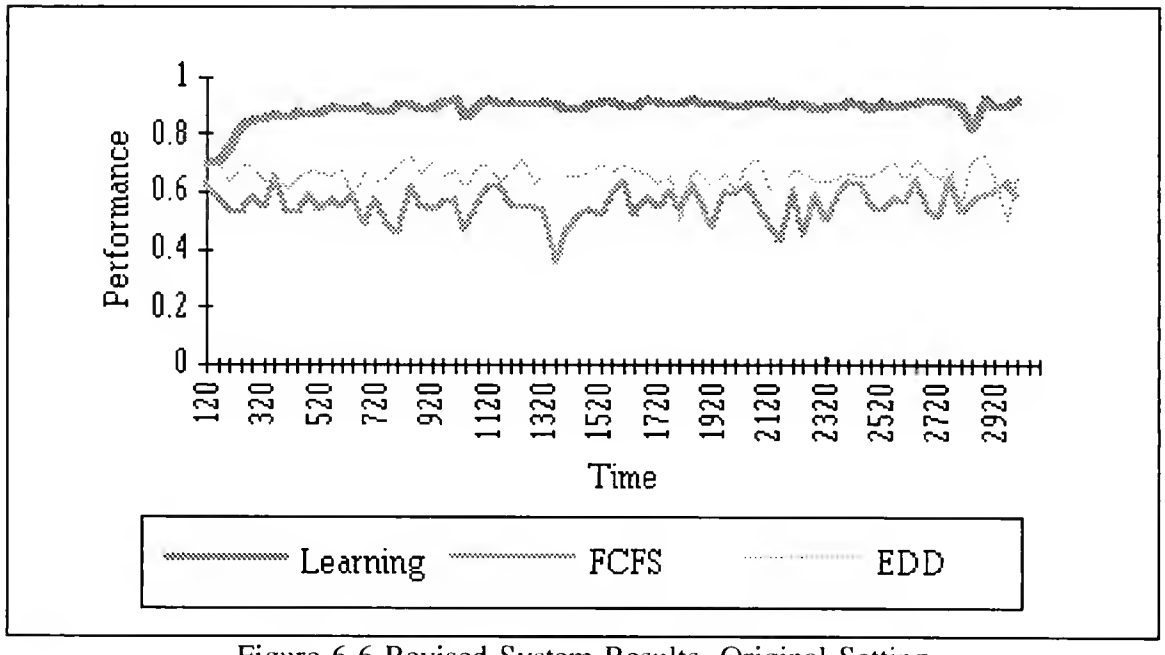

Figure 6-6 Revised System Results--Original Setting

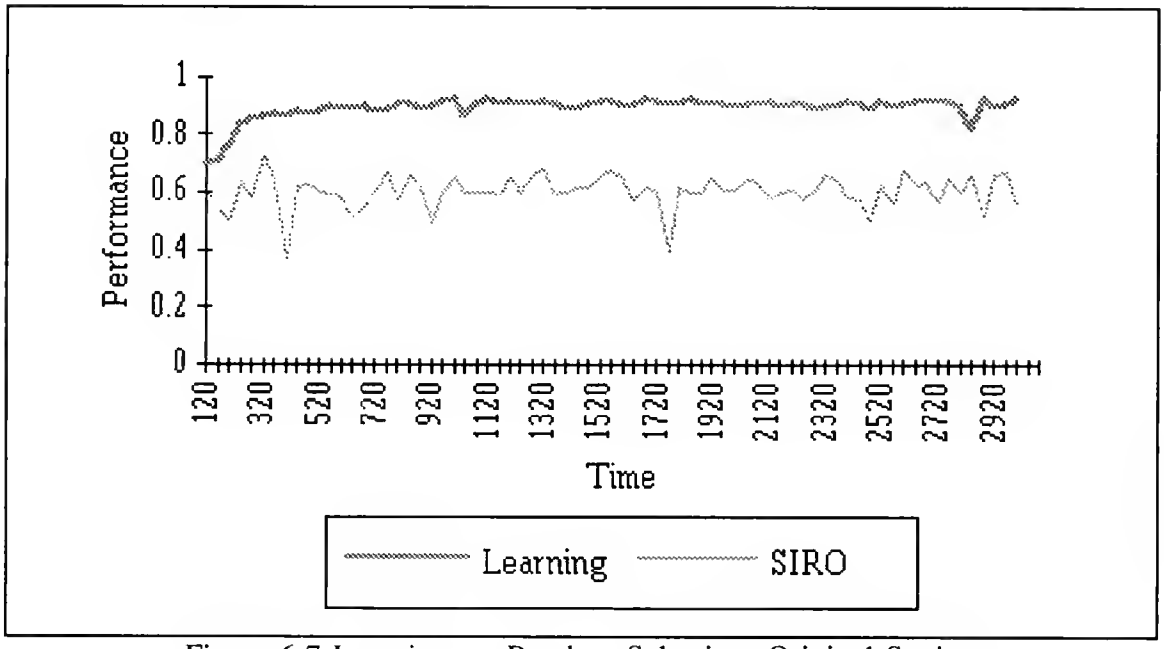

Figure 6-7 Learning vs. Random Selection--Original Setting

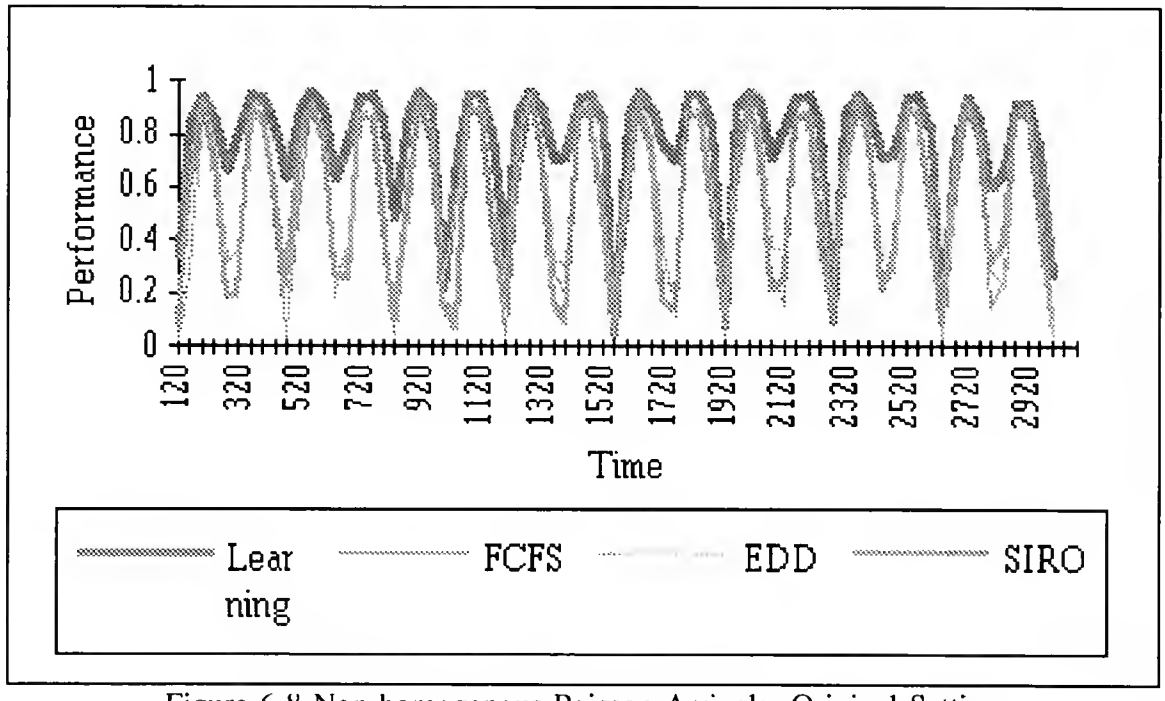

Figure 6-8 Non-homogenous Poisson Arrivals--Original Setting

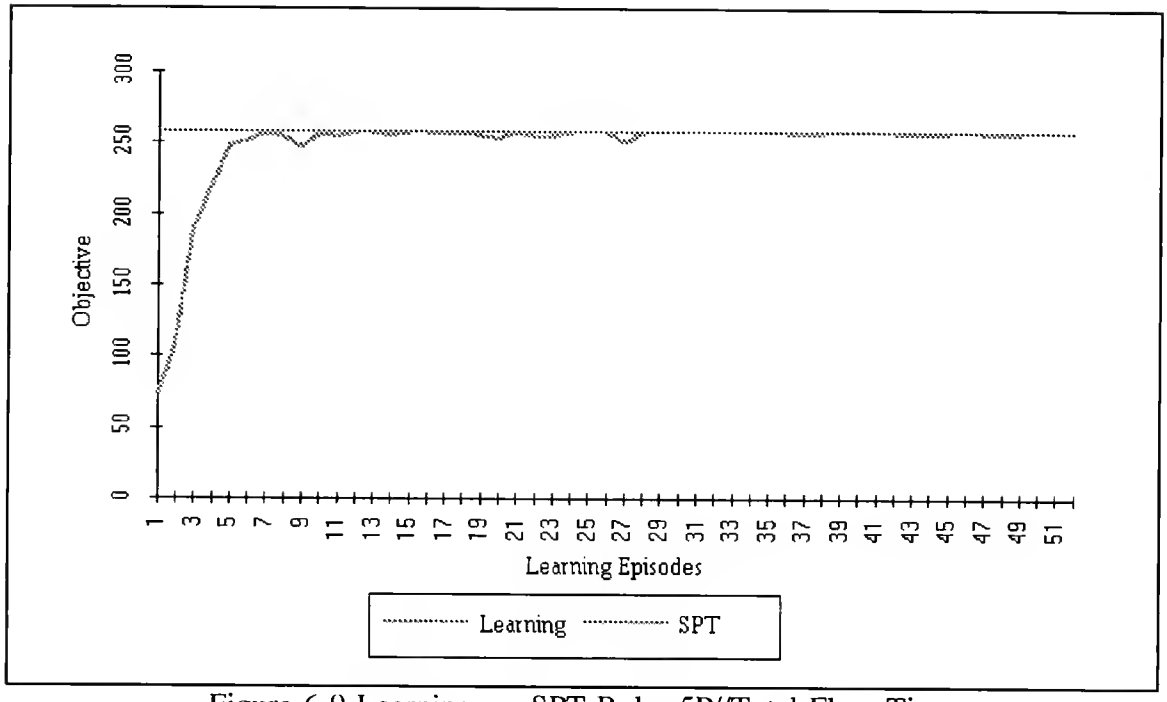

Figure 6-9 Learning vs. SPT Rule--5P//Total Flow Time

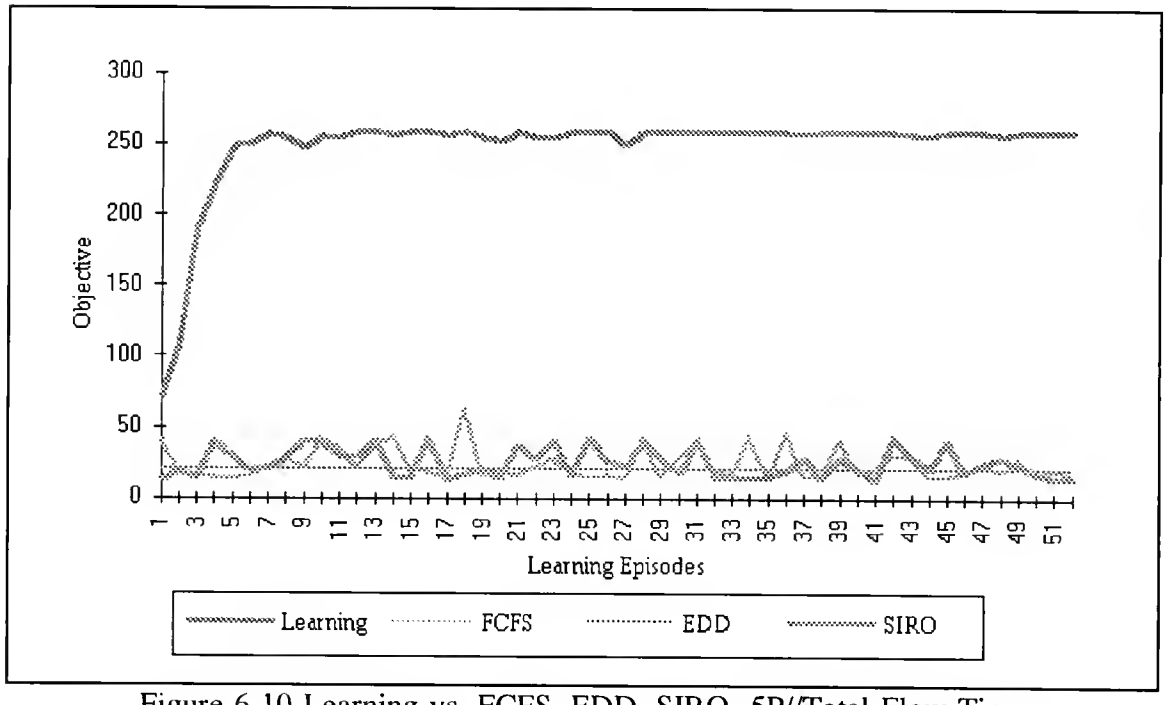

Figure 6-10 Learning vs. FCFS, EDD, SIRO--5P//Total Flow Time

CHAPTER 7
GENERALIZATION OF THE GENETIC LEARNING SYSTEM

7.1 Overview

The system described in Chapter 6 has many limitations on the generality and the expressive power of the rule language used. In this chapter we generalize the simulation environment to a flowshop setting and also generalize the rule language so as to capture more information about the environment.

7.2 The Flowshop Extension

In our terminology a flowshop is defined as a series of machines lined up in sequence. A job has to go through each machine in this predetermined order before it is finished. Each machine can process one job at a time and may have arbitrary processing times. In front of each machine a queue of limited size serves as the Work In Progress (WIP) inventory. Each queue is controlled by a dispatcher who possesses some knowledge of the environment. Jobs are dispatched according to the dispatcher's knowledge at the time of dispatch. After a job is completed it is evaluated with respect to the performance measure specified and this information is sent back to the dispatcher to use for updating its knowledge base. Figure 7-1 summarizes this idea.

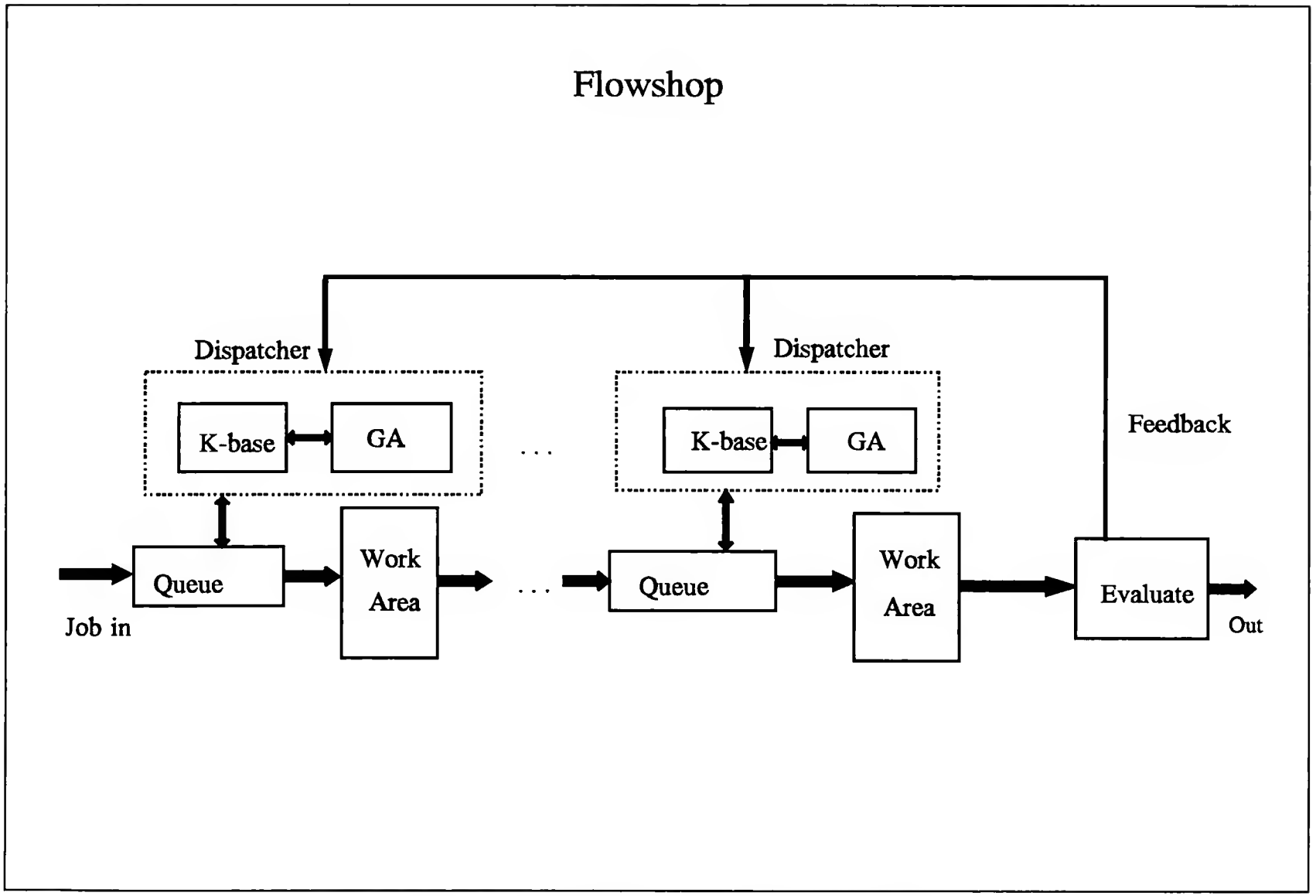

Figure 7-1 A Flowshop with Intelligent Dispatchers

7.3. Knowledge Representation

A knowledge representation scheme has been developed to incorporate different aspects of scheduling knowledge. Each rule in this representation corresponds to using conjunctions of dispatching rules subject to the existence of certain conditions in the environment. The basic language primitives are: Atoms, Predicates, and Condition-Action Rules. Rules are conjunctions of these primitives.

Attributes are used to capture information about the factory floor status as well as the descriptions of jobs currently awaiting for processing. It is also possible to define functions of attributes which will be treated as regular attributes. Domain of each attribute has to be defined before it can be used. Each attribute can take on values from the set of integers and real numbers as well as nominal valued sets. An atom, defined over an attribute is the constraint defined on the domain of the attribute. For example, an atom for a nominal attribute like "Priority" may be (Priority = Urgent); and for a real valued attribute like "Processing-Time", an atom may specify (5.0 $\leq$ Processing-Time $\leq$ 8.5.) Such atoms are referred to as Simple Atoms. Compound atoms are comprised of disjunction of Simple Atoms of the same attribute. For example, (Processing-Time $\leq$ 1.3 $\vee$ Processing-Time $\geq$ 1.6) is a Compound Atom where "$\vee$" is the "logical-or" operator.

Certain attributes may be job-related, in the sense that they may evaluate to different values for different jobs in the queues. The evaluation of an atom of such an attribute defines a subset of jobs whose related attribute values are in the current value set of the atom. Atoms that are not job-related select the current set of the jobs as the

subset. They do not distinguish among jobs, rather they look at whether certain environment parameters are satisfied or not.

A "Predicate" is a function that selects a subset of the current jobs with respect to its selection criteria. Standard dispatching rules like, SPT, EDD, etc. are modelled as predicates. Though simple dispatching rules do not select more than one job (except for ties), Predicates retain the flexibility for representing other more complex algorithms for dispatching.

Traditional "Production Rules" or "IF-THEN rules" called CA_rules (in our terminology) are used to model various known scheduling heuristics. These may be strategies that human experts have found useful or heuristics published in literature known to work fine for some environments.

All conjuncts return a truth value which may take values from the set "{True, False, Unknown}. An atom evaluates to True if the current system attribute value is in the value set of the Atom. Predicates always return true since they can always select a job given that the queue is not empty. A CA_rule evaluates to True if its condition part evaluates to True.

7.3.1 BNF Form of the Rule Language

[ ]: any parameter listed in the square brackets is optional.

{ }: any parameter listed in the braces has to be specified.

truth_value: one of

true, false, unknown

rule:

AND (conjunct$_i$)

conjunct:

atom

predicate

ca_rule

atom:

simple_atom

compound_atom

compound atom:

OR(simple_atom$_i$)

simple_atom:

(value$_1$ <= linear_attribute <= value$_2$)

(nominal_attribute = value)

predicate:

$f: X \rightarrow Y$ where $Y \subseteq X$ predicate is a function that selects a subset from a given set.

ca_rule:

conditions

logical_expression

actions

predicate

logical_expression:

atom

[logical_unary_opr]atom [{logical_binary_opr} logical_expression]

logical_opr:

logical_unary opr: one of

!(NOT)

logical_binary_opr: one of

&(AND), /(OR)

value:

number

symbol

attribute:

linear_attribute

nominal_attribute

function_of(attribute)

function_of(attribute):

$f: D \rightarrow X$ where D is the domain of the attribute and X is some nominal or continuous valued set.

7.3.2 Rule Level Recombination Operators

Four operators are defined for genetic recombination: crossover (X), mutation (μ), generalization (G) and specialization (S). Crossover and mutation are rule level operators,

i.e., they operate on entire rules. Generalization and specialization are defined to operate on certain conjuncts within rules. A conjunct level mutation operator is also defined. All these operators, with the exception of mutation, are based on the fitness-values of the conjuncts in the rules considered for recombination.

Crossover. This is an adaptation of the uniform crossover operator used with binary coded GAs. Given two parents, different Conjuncts are probabilistically selected for crossover based on their fitness values, with higher fitness Conjuncts having greater probability of participating in the crossover. For each Rule, the number of selections is approximately half the number conjuncts in the rule (each selection is made probabilistically, based on the conjunct fitness values, and thus the same conjunct may be selected more than once).

Consider the parents:

R_i: $C_{1i} \wedge \underline{C}_{2i} \wedge \underline{C}_{3i} \wedge C_{4i}$ (C_{2i} and C_{3i} chosen for Crossover)

R_j: $\underline{C}_{1j} \wedge C_{2j} \wedge \underline{C}_{3j} \wedge C_{4j}$ (C_{1j} and C_{3j} chosen for Crossover)

Children may be formed in two ways:

(1) $Child_i$: $C_{1i} \wedge C_{1j} \wedge C_{3j} \wedge C_{4i}$

$Child_j$: $C_{2i} \wedge C_{2j} \wedge C_{3i} \wedge C_{4j}$

Here, $Child_i$ is obtained from $Rule_i$, after replacing the selected conjuncts with those from the other parent. $Child_j$ is similarly obtained.

(2) $Child_1$: $C_{1j} \wedge C_{2i} \wedge C_{3i} \wedge C_{3j}$

$Child_2$: $C_{1i} \wedge C_{2j} \wedge C_{4i} \wedge C_{4j}$

Here, $Child_1$ is formed by combining the chosen conjuncts from both parents. This rule thus now contains the "best" conjuncts from both parents. The second child, containing the weaker conjuncts, may be discarded. In our system we have implemented the second crossover operator described.

Mutation. Mutation at the rule-level, occurs through the random replacing of one conjunct by another of the same type.

7.3.3 Conjunct-level operators

Recombination operators are used to change the current value of the conjuncts if the conjunct type is suitable. Of the defined conjunct types, only atoms are considered for recombination. Predicates, being standard dispatching rules or other job-selection routines, do not have value assignments that may be used in recombination. CA_rules too, are used to incorporate heuristics that a human dispatcher may use, and so no genetic operators are applied.

Let v be an attribute. Let $V = \{ v_1, ..., v_n \}$ be the set of values that it can take. Let a be an atom defined on that attribute. Let $A = \{ a_1, ..., a_k \}$ $A \subseteq V$ be the current binding of the atom a. Let $|V| = n$ and $|A| = k$..

Let $U(1,k)$, $k \in \mathbb{N}$, be a discrete uniform distribution, that returns integers between 1 and k.

Let $U(a,b)$, $a,b \in \mathbb{R}$, be a continuous uniform distribution, that returns real numbers between a and b.

If the attribute is nominal it will be assumed that v is a set of nominal valued elements and so is a. If the attribute is continuous then it will be assumed that $v_i = [v_{i1},v_{i2}]$ where $v_{i1},v_{i2} \in \mathbb{R}$ and so is a_i.

Mutation. Mutation is one of the random operators that is used to "jump start" the current population. Here in this context mutation operator randomly selects an element in the current binding and alters it according to the type of the element. More formally it is defined as follows.

Let $i = U(1,k)$, $k \in \mathbb{N}$. Let $a_i \in A$, then $\mu(a) = (A-\{a_i\} \cup \{v_j\}$ where $j=U(1,n)$ and $v_j \in V$.

Example 1. Let queue_length be an attribute. Let V = { [0,100] }. Let queue_length_q1 be an atom defined on queue_length. Let A = { [3,7], [9,11] }. If we consider this as part of a production rule then it reads as

if $2 \leq$ queue_length_q1 $\leq 7 \vee 7 \leq$ queue_length_q1 ≤ 11.

Mutation operator μ(queue_length_q1) will pick one of the elements in A and alter it randomly by checking the domain. Assume [3,5] is picked to be changed and assume new lower and upper bounds are 2 and 9. Then the resulting set A is { [2,9] } U { [9,11] }, which equals { [2,11] }.

Example 2. Let server_rate be an attribute. Let $V = \{ 100,150,200,250 \}$. Let server_rate_s1 be nominal atom defined on server_rate. Let $A = \{ 100,150 \}$. Again as a production rule this reads as

if server_rate_s1 = 100 $\vee$ server_rate_s1 = 150.

Then μ(server_rate_s1) will pick one of the elements in A and alter it according to the uniform distribution defined on the domain. Assume 150 is selected to be changed and assume 200 is selected from the domain. Then set A = {100}∪{200} = {100,200}.

Generalization. Generalization is performed when the strength of the atom is less then a specified threshold. Note that a low strength can occur if the atom is too specific to match any state seen so far. A generalization will relax the temporal constraint imposed on the atom.

Let a be continuous, then $G(a) = A - \{a_i\} + v_i'$, where $v_i' \supset a_i$ and $v_i' \subseteq V$. Let a be nominal, then $G(a) = A + \{v_i\}$, where $v_i \notin A$.

Example 3. Consider the same attribute of Example 1 of mutation, then G(queue_length_q1) will pick one of the elements and make its upper and lower bound strictly looser. The rest is exactly the same as the mutation element. In the nominal case, a new element from the domain will be added to the set A.

Specialization. Contrary to the situation mentioned above, sometimes the strength of an atom may be too high which is possible when the constraints on the atom are too loose, i.e. it is too general. This situation will lead to over generalization which will, on the long run deteriorate the average system performance, i.e. it will produce bad rules that fire all the time. To avoid this situation the specialization operator is used if the strength is higher than a threshold.

Let v be continuous, then $G(a) = A - \{a_i\} + v_i'$, where $v_i' \subset a_i$. Let v be nominal, then $G(a) = A - \{a_i\}$, where $a_i \in A$.

Example 4. Consider the attribute mentioned in example 2 of mutation, then S(server_rate_s1) will take out one of the elements in set A. In the continuous case an element will be picked randomly and the upper and lower bounds will be made strictly tighter.

7.3.4 Inference Strategies

Consider a rule R_j as:

R_j: C_{1j} AND C_{2j} AND ... AND C_{nj}. (Each C_{ij} here is called a conjunct.)

In order to formalize the inference strategy, we define the following:

Definition. Rule R_j is *active* if all C_{ij} evaluate to True, i.e., if evaluate(C_{ij}) = TRUE for all i=1 to n.

Definition 7.1: Rule R_j is *consistent* if it is active and there is at least one common job voted for by all the conjuncts constituting R_j, i.e., if evaluate(C_{ij}) = TRUE for all i=1 to n, and AND (V_{ij}) != NULL, where V_{ij} = subset of jobs voted for by conjunct C_{ij}, and NULL is the null set.

Definition 7.2: A rule gives *actual votes* if it is consistent. All jobs voted for by this rule receive actual votes.

Definition 7.3: A conjunct always gives *virtual votes* if it evaluates to true. Note that even though Rule R_j may not be active, any Conjunct C_{ij} can give virtual votes to jobs as long as evaluate(C_{ij}) = TRUE.

The evaluation of a rule proceeds as follows. First the atoms in the rule are evaluated and intersection of their votes is found. If that is null than the rule is not

consistent. If that intersection is not null then Predicates and CA_rules are evaluated by using this constrained set.

Example 5. Assume there are five jobs waiting in a particular queue and one is to be dispatched. Assume $Rule_i$ has two Atoms and one Predicate and one CA_rule. Suppose $Atom_1$ votes for jobs 1,2 and 3, $Atom_2$ votes for jobs 2,3 and 4. Clearly their intersection consists of jobs 2 and 3. This rule will be consistent only if the Predicate and the CA_rule vote on the same job out of jobs 2 and 3.

Inferencing is done by the dispatcher and each dispatcher has a knowledge base of rules relevant to dispatching. All jobs currently in the queue are possible candidates for dispatch.

Inference (that is, selection of a job for dispatch) proceeds through the evaluation of all the rules in the knowledge base of the concerned dispatcher. In evaluating a rule, all the conjuncts in the rule are evaluated. If a conjunct evaluates to TRUE, it yields a set of jobs with virtual votes. A rule, if found consistent after evaluation, also yields a set of jobs with actual votes.

The cumulative vote distribution of the jobs to be dispatched, is generated at that particular decision point, by using actual votes, if there exists any. If there are no consistent rules, i.e. no actual votes, the aforementioned distribution is generated by using virtual votes. If there are no virtual votes, then a discrete uniform distribution is used as the vote distribution. After the cumulative vote distribution is generated a job is selected randomly by using that distribution. Basically the more votes a job gets, the more are the odds that it is going to be selected for processing.

7.3.5 Credit Assignment

Let $F(C_{ij})$ be the fitness of the i^{th} Conjunct C_{ij} in $Rule_j$.

We define Overall Rule Fitness $F(R_j)$ as:

$$F(R_j) = [\sum_i F(C_{ij}) w_i] w + (1 - w) F, \quad (1)$$

where w_i comes from a weight function (currently all equal), w is the importance given to the average conjunct fitnesses and F is the average strength acquired by the rule through issuing actual votes.

Partial Credit for Conjuncts. Since conjuncts may give Virtual Votes, even when the associated rule is not active, partial credit is assigned to individual conjuncts. Here, since the rule is not active, the rule fitness $F(R_j)$ should not increase proportionately with the increase in any $F(C_{ij})$. The following compensation scheme is proposed:

$\Delta F(C_{ij}) = f(\text{virtual votes})$ for some function f which may be defined as:

$$f_{obj} \cdot k \cdot \frac{\#\text{ of virtual votes assigned by } C_{ij} \text{ to the selcted job}}{\text{total \# of virtual votes received by the selected job}}$$

where f_{obj} is the payoff received by the selected job. One strategy may be give the highest credit to a consistent rule, somewhat lower for the active but inconsistent rule and the lowest for the inactive rule. Below is an example of this strategy.

$$k = \begin{cases} .7 \text{ if } R_j \text{ is consistent} \\ .5 \text{ if } R_j \text{ is active but not consistent} \\ .3 \text{ if } R_j \text{ is inactive} \end{cases}$$

7.4 Simulation Experiments

In this section we will investigate how the learning system performs within a relatively simple flowshop environment. We will discuss the shortcomings of the system and some explanation of how to improve these shortcomings.

Before we proceed with describing the simulations we should explain the naming conventions about a particular simulation run. Since we do not have enough space in the figures, results of each simulation run is encoded. All figures are displayed at the end of this chapter. For a non-learning simulation we named the related graph with the name of the dispatching rule used, such as SPT, FCFS, etc. Results of learning system is encoded as "9999.9.xxxx.xx" where a "9" refers to a digit and an "x" refers to a character. The first digit holds 10 times the weight (w) used in equation (1). The second digit shows 10 times the crossover rate used. The third and the fourth digit holds 100 times the mutation rate used. The digit after the dot holds 10 times the constant k described in equation (2). The first set of characters explain whether intermediate evaluation points and a certain voting scheme is used or not. The second set of characters explain if the degenerate rules were removed (i.e., a "d" is present) and if fitness based voting is used (i.e., a "p" is present.)

7.4.1 Experiment Set One

In this set of experiments we have looked at a two machine flowshop, where jobs arrive according to Poisson process with mean 50. The processing times of jobs are drawn from a uniform distribution between 0.01 and 0.019 for Machine 1 and between

0.05 and 0.025 for machine two. The initial knowledge base had 15 rules and 5 conjuncts per rule created randomly. The objective function we tried to maximize was the percentage of time a job spends for service, hence minimizing the percentage of time it spends waiting. Each learning episode was run with an average of thousand jobs. The simulation was run for hundred and fifty learning episodes. To compare the effect of learning, the system was also run by using simple dispatching rules including, FCFS, SPT, EDD and RNDM. Figure 7-2 displays the performance of the learning system compared to the systems with only simple dispatching rules.

Figure 7-2 clearly shows that the learning system is performing better than FCFS and as good as SPT. So this suggests that the resulting knowledge base should be dominated by SPT or equivalent rules. On the contrary the resulting knowledge base has converged to LPT and LRPT. The explanation is quite interesting. One way to maximize the percentage of time spent for service is that the system should use SPT so that the remaining jobs wait less and their waiting times are shorter when compared to using LPT. But this requires a more than one step deduction which the system is unable to do yet. Instead the system, since it is evaluated on a per job basis selects the job that has the longest processing time so that it can maximize the percentage of time it spent in the queue. After all it is being evaluated only on that job, choosing SPT will actually minimize the time it has spent for processing. Basically this objective function can be interpreted as maximizing the dollar value of a job, since most of the time jobs that require more resources will have more dollar value from a cost accounting point of view. So the system does the right thing and chooses the jobs with the highest dollar value first.

This is consistent with the results presented in Blackstone, Phillips and Hogg (1982) that the value related dispatching is basically using LPT or LRPT and will result in high WIP and reduced throughput.

7.4.2 Experiment Set Two

The first set of experiments revealed the fact that when the knowledge base "converged," most of the rules consisted of multiple copies of the same conjuncts and had short lengths (i.e. number of conjuncts per rule were around two or three.) By taking this into consideration we changed the rules such that each had at most three conjuncts. To compensate for the number of conjuncts lost per knowledge base the knowledge base size was increased to twenty.

In this set of experiments we have looked at a two machine flowshop, where jobs arrive according to Poisson process with an arrival rate of 0.5 jobs per unit time. The processing times of jobs are drawn from a uniform distribution between 0.1 and 1.9 for Machine 1 and between .5 and 2.5 for machine two. The initial knowledge base had twenty rules and three conjuncts per rule, created randomly. The objective function we tried to maximize was

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{(F_i)^2},$$

where F_i is the flow time of job i and n is the number of jobs dispatched. The idea was to minimize the average flow time of jobs per learning episode. Each learning episode consisted of an average of thousand jobs. The simulation was run for two hundred

learning episodes. To compare the effect of learning, the system was also run by using simple dispatching rules including, FCFS, SPT, EDD and RNDM. Figure 7-3 displays the performance of the learning system compared to the systems with only simple dispatching rules. Mean waiting time in Queue 1 and mean waiting time in Queue 2 are displayed in Figures 7-4 and 7-5 respectively.

7.4.3 Experiment Set Three

Realizing that differences in the performance of the best rule (SPT) and the worst (FCFS or EDD) was too small in the second set of experiments, we increased the two parameters of the simulation setting. We first changed the coefficient of variation of the processing time assignment distribution from .27 to .43 and increased the traffic intensity factor from .5 to .75 hoping that the differences among the performances of the best and the worst would increase giving more chance for improvement for the learning system.

Another realization from additional runs revealed the fact that increasing the coefficient of variation for processing time assignments gave the learning system more ability to differentiate between good and bad rules with more consistency. With low coefficient of variation the probability that a bad rule is favored was very close to the probability that a good rule is favored. This is analogous to the Type II error when comparing two populations.

In this set of experiments we have looked at a two machine flowshop, where jobs arrive according to Poisson process with an arrival rate of 0.5 jobs per unit time. The processing times of jobs are drawn from a uniform distribution between 0.1 and 2.9 for

Machine 1 and between 0.5 and 3.1 for machine two, thus increasing the coefficient of variation of the processing times and the traffic intensity factor. The initial knowledge base had twenty rules and three conjuncts per rule created randomly. The objective function we tried to maximize was

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{(F_i)^2},$$

where F_i is the flow time of job i and n is the number of jobs dispatched. The idea was to minimize the average flow time of jobs per learning episode. The simulation was run for two hundred learning episodes with thousand jobs passing through the system per learning episode. To compare the effect of learning, the system was also run by using simple dispatching rules including, FCFS, SPT, EDD and RNDM. Figure 7-6 displays the performance of the learning system compared to the systems with only simple dispatching rules. Mean waiting time in Queue 1 and mean waiting time in Queue 2 are displayed in Figures 7-7 and 7-8 respectively.

7.5 Interpretation of the Simulation Results

Simulation experiments showed that the system is able to discover the usefulness of certain dispatching rules but was unable to utilize the environment related attributes like server utilization and queue lengths. Some job related attributes were tried but since they were irrelevant to the objective function they either disappeared from existence very quickly or generated "degenerate rules." One reason for not being able to discover useful conjuncts related to system attributes is due to the fact that, since some may have

unbounded and continuous domains it may take the system longer than we allow to discover such useful ranges. For example, the system may find a rule that reads as " if the queue length is greater than fifty use SPT," whereas the system never reaches a queue length of fifty throughout the simulation. There is no feedback to the rule as to how to direct its search for a better queue length related atom. Mutation is the only search mechanism of the system to discover such an atom.

One other problem discovered is the degeneracy of the knowledge base. Certain attributes are allowed to vote for all the jobs if they evaluate to true. Given that within a learning episode they evaluate to true all the time they gather the maximum fitness possible since no matter what job is chosen they get a credit back. After a couple of learning events such atoms begin to dominate the knowledge base where the conjuncts that are responsible for the actual action disappear since they do not vote all the time for all the jobs. Quite a large number of simulations converged to such a knowledge base where the system behavior is "random."

As it is, the crossover operator in use is very focused. Since we discard the low performing conjuncts, after crossover we have no means of getting the parents back if the new rule turns out to be a useless one. As it is the crossover operator is working as a random "trial and error" operator which moves the population to a local maxima and homogenizing the population very early in the search. Some results shows a rapid performance improvement in the system which may persist for a long time (these are actually our very best results) but most of the time it converges to a knowledge base which is actually irrelevant to the performance major and usually degenerate.

The number of conjuncts is another possible problem. When there is one measure of performance it may be beneficial to keep the rules short. Whereas if the performance measure is a function of many factors it may be beneficial to increase the number of conjuncts. But in the existence of dispatching rules long rules are bound to die since the dispatching rules will almost never agree on a job. Earlier experience described in Chapter 6 had shown that the knowledge base may not select a job around fifty percent of the time. This may be avoided by relaxing the selection criteria of the dispatching functions. Instead of selecting a single job dispatching functions can be designed such that they select a subset which satisfies one selection criteria that is not necessarily the maximum or minimum. Furthermore, dispatching functions can be replaced by job related attributes.

To see the effect of degeneracy we removed the degenerate attributes and dispatching rules from the domain. As a result the system performance improved noticeably. One other concern was to see how fitness based voting would effect the system. Enabling fitness proportional voting improved the performance. The system behavior under these conditions is summarized in Figures 7-9, 7-10,7-11. Below is a summary of a paired t test. Table 7-1 summarizes the t statistic for paired differences and the corresponding p-values are displayed in Table 7-2. The t statistic is defined as

$$\frac{(r_i - c_j)\sqrt{n}}{s(r_i - c_j)}$$

where r_i and c_j are the mean values of the outputs for the experiments listed in the i^{th} row and the j^{th} column of the tables.

Table 7-1 Paired t Statistics

	3705.5.nav	4705.7.anv	4505.7.nav.dp	fcfs	spt
3520.nanv.d	20.655	3.085	15.780	85.423	-8.515
3705.5.nav		-14.486	-9.252	69.426	-25.364
4705.7.anv			8.131	73.716	-9.927
4505.7.nav.dp				77.716	-27.348

Table 7-2 p Values

	3705.5.nav	4705.7.anv	4505.7.nav.dp	fcfs	spt
3520.nanv.d	5.7E-38	0.00132	4.5E-29	7.5E-95	9.3E-14
3705.5.nav		1.7E-26	2.34E-15	4.4E-86	2.1E-45
4705.7.anv			6.3E-13	1.7E-88	7.9E-17
4505.7.nav.dp				7.6E-91	3.1E-48

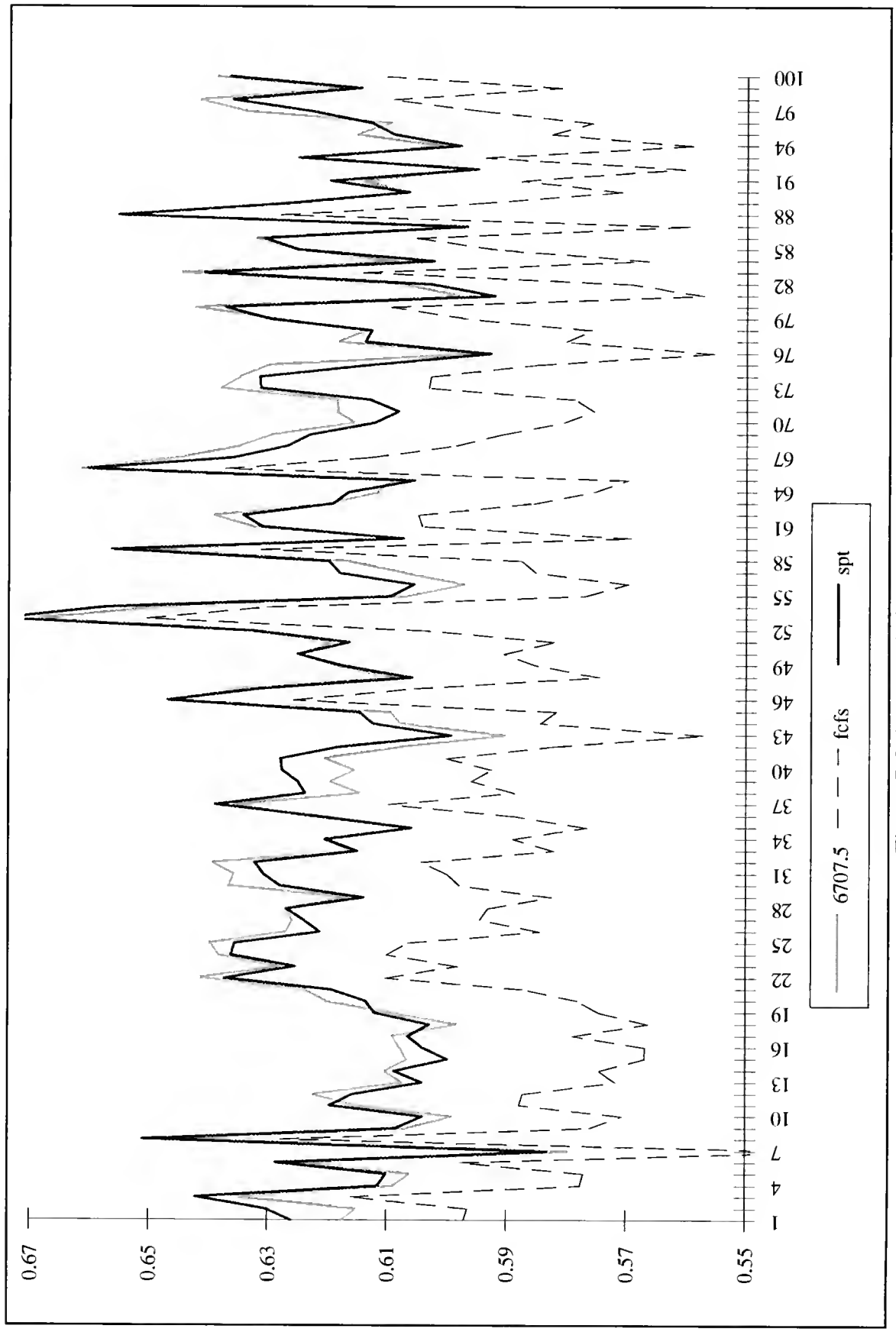

Figure 7-2 Average Objective--Set 1

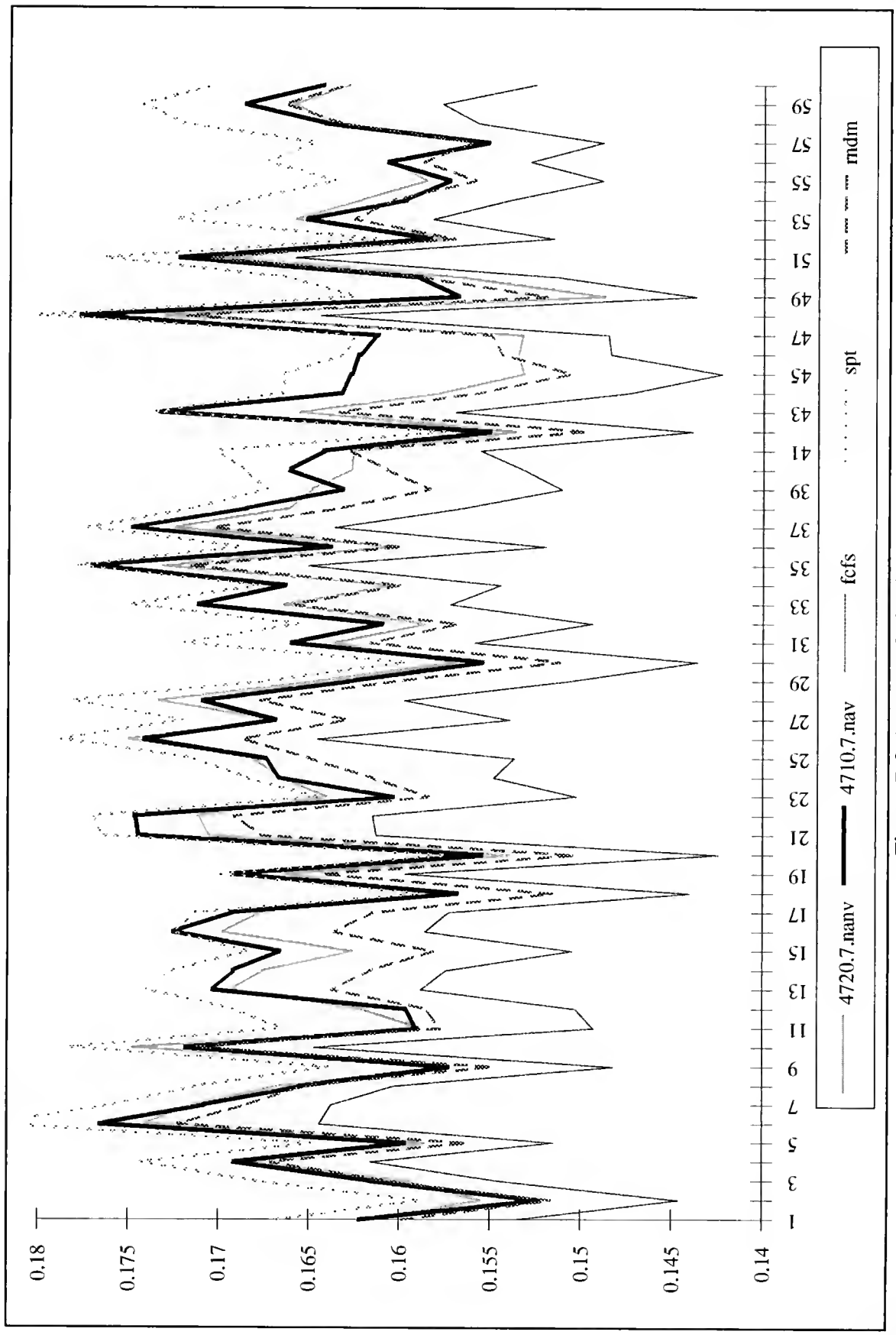

Figure 7-3 Average Objective--Set 2

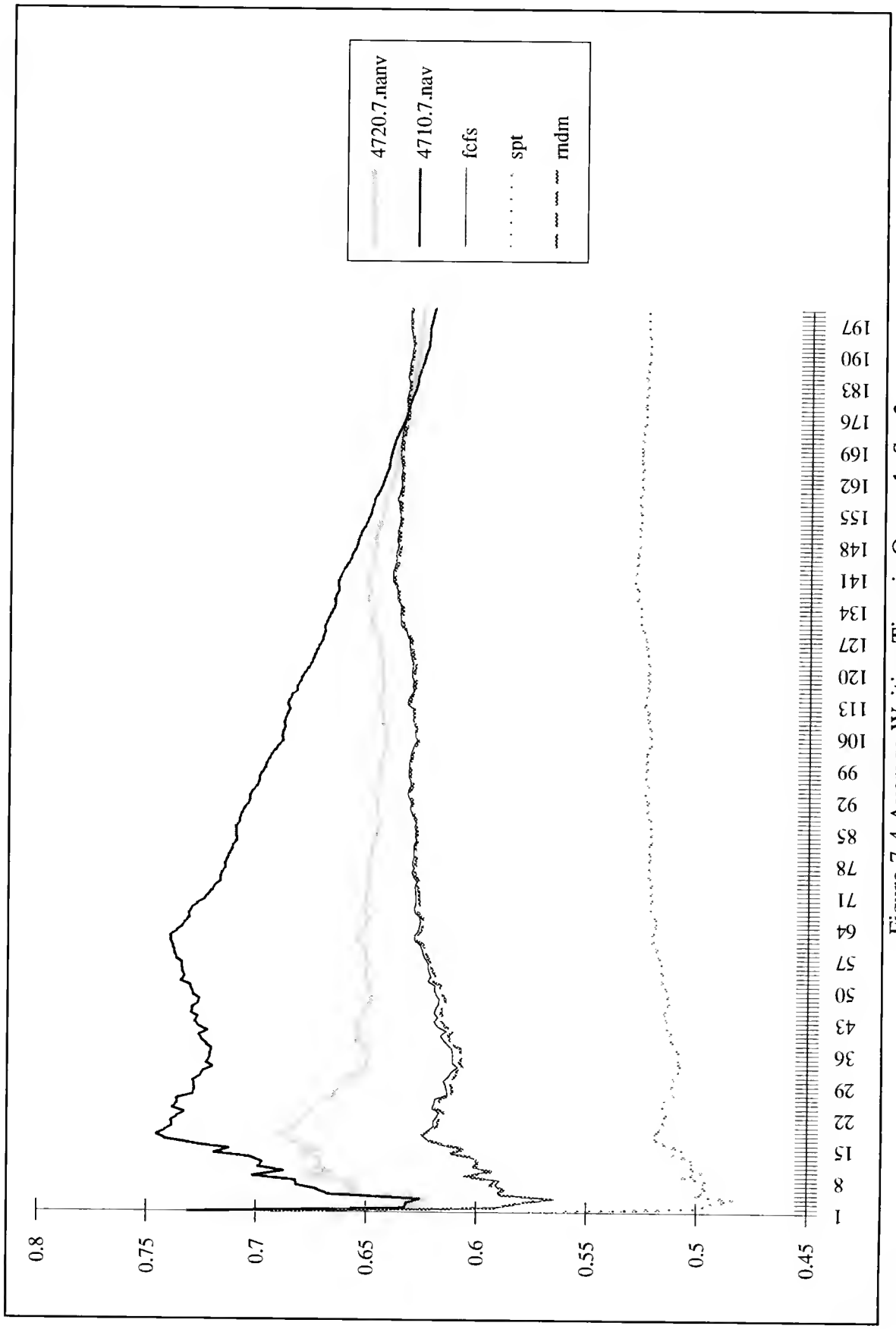

Figure 7-4 Average Waiting Time in Queue 1--Set 2

Figure 7-5 Average Waiting Time in Queue 2--Set 2

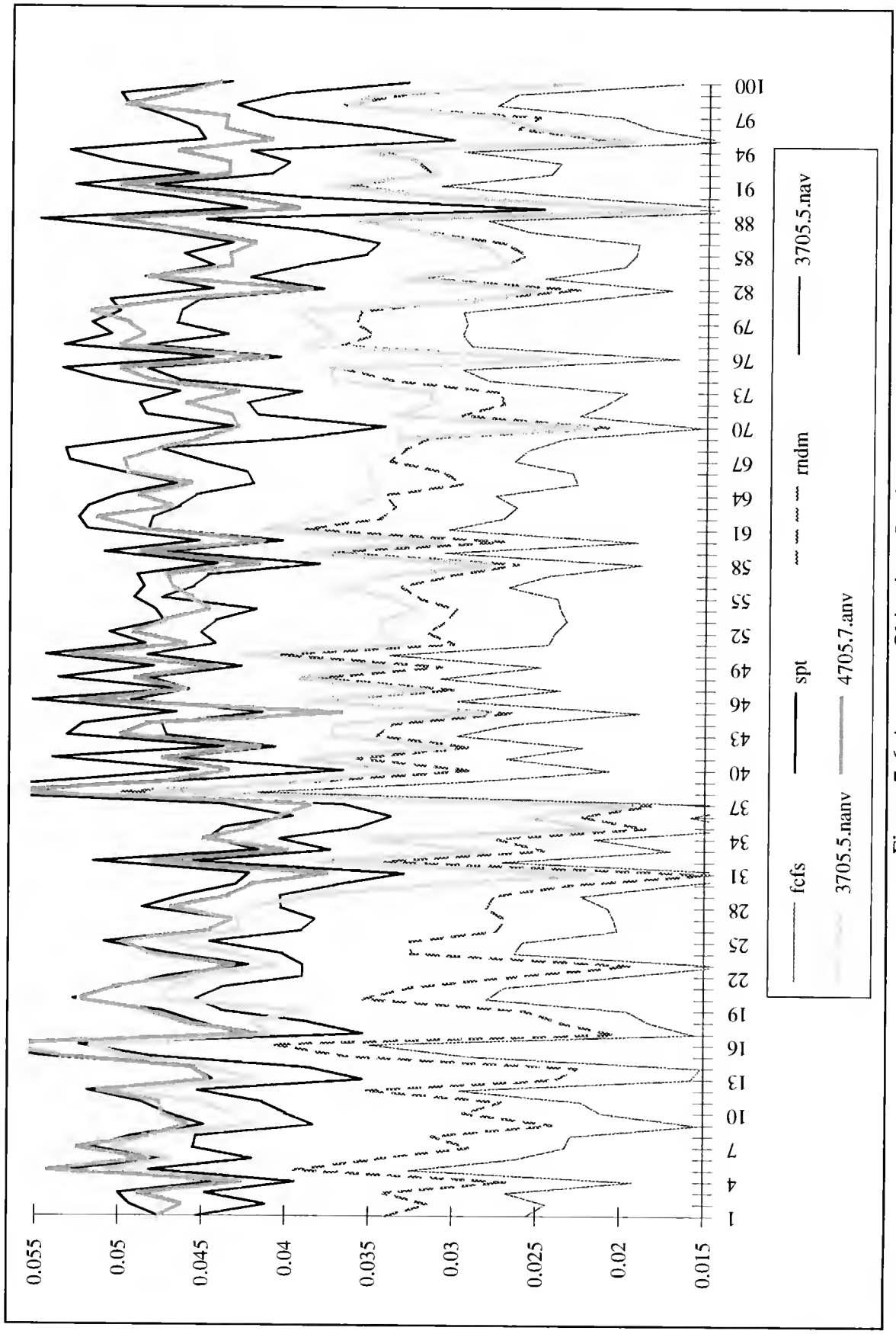

Figure 7-6 Average Objective--Set 3

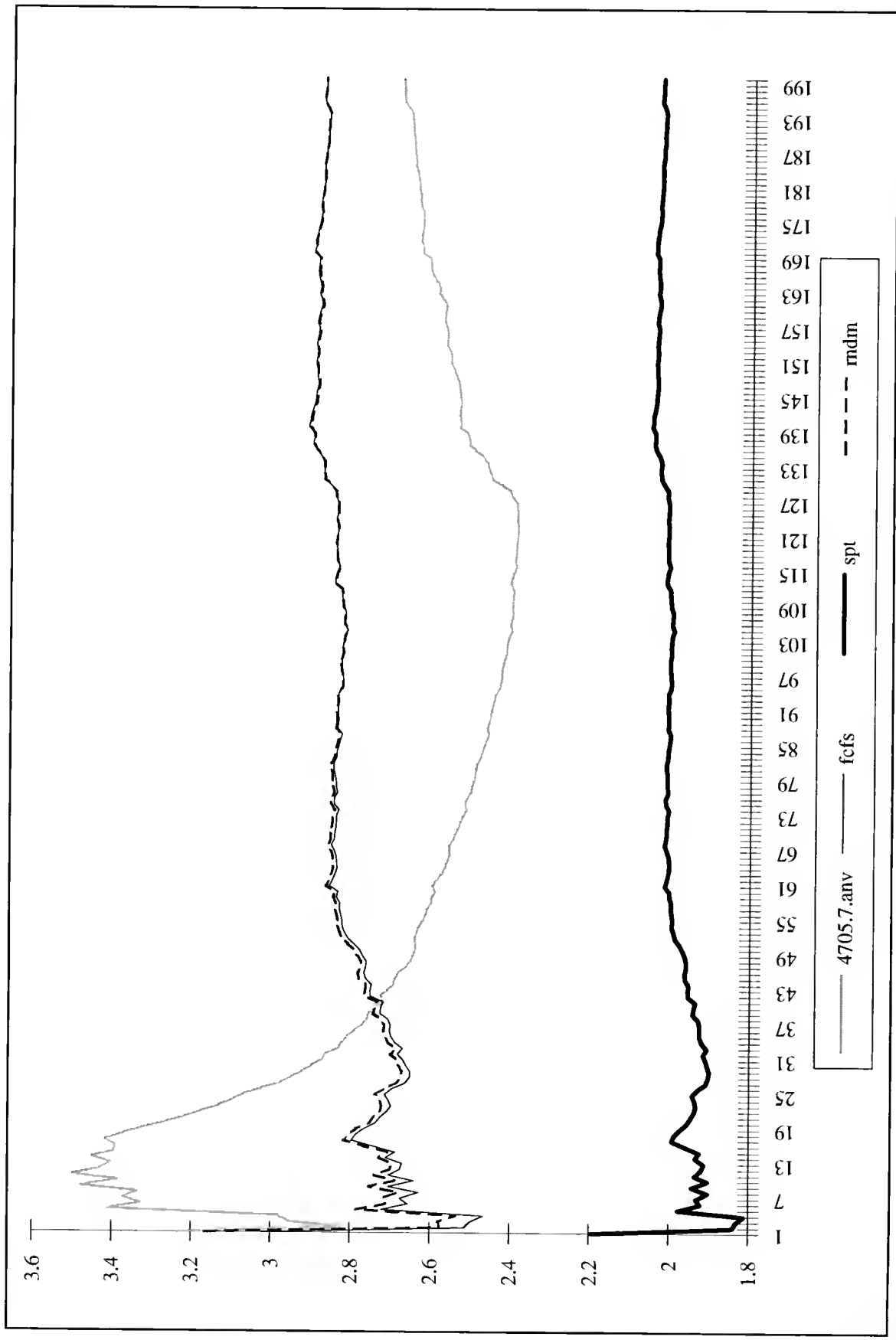

Figure 7-7 Average Waiting Time in Queue 1--Set 3

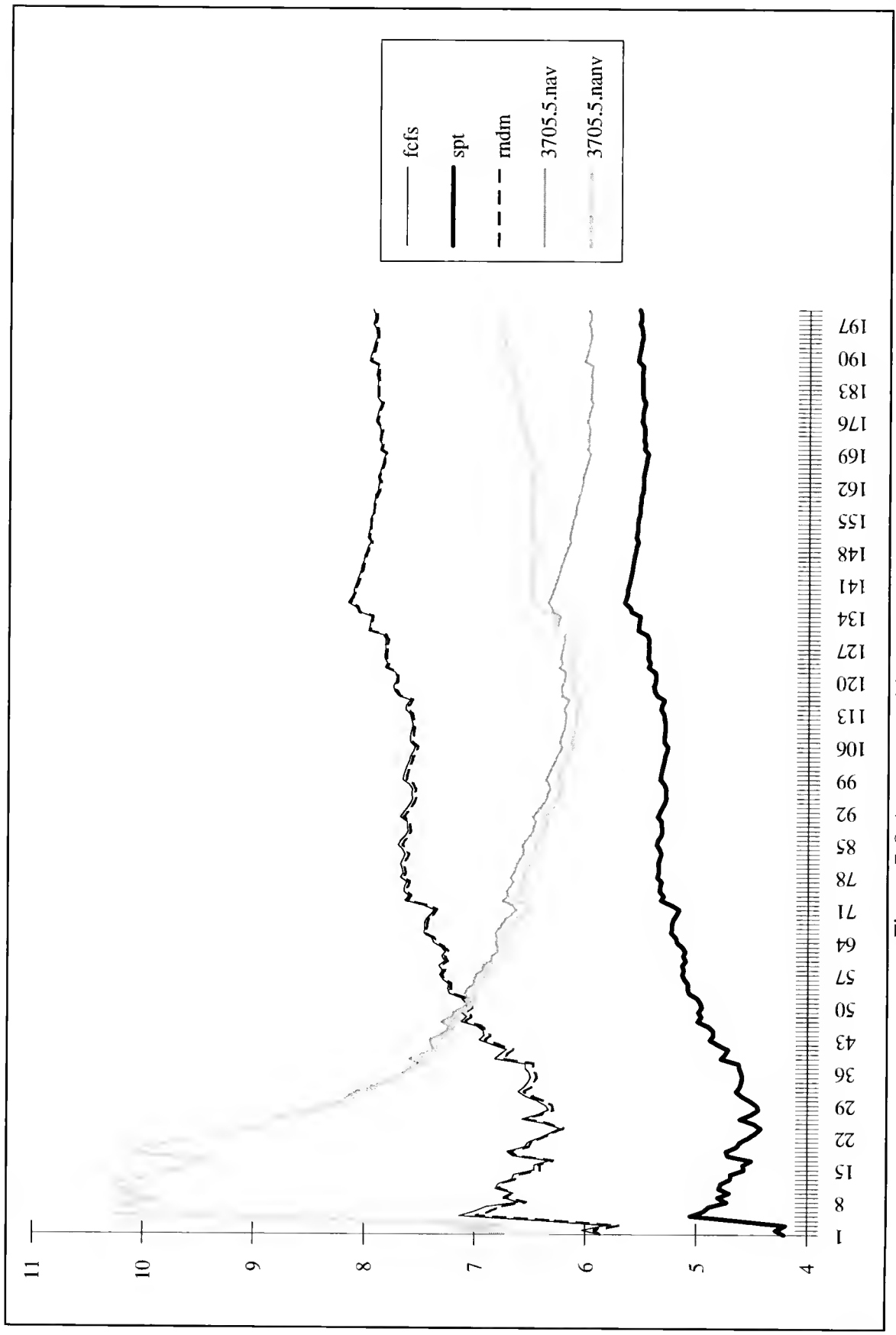

Figure 7-8 Average Waiting Time in Queue 2--Set 3

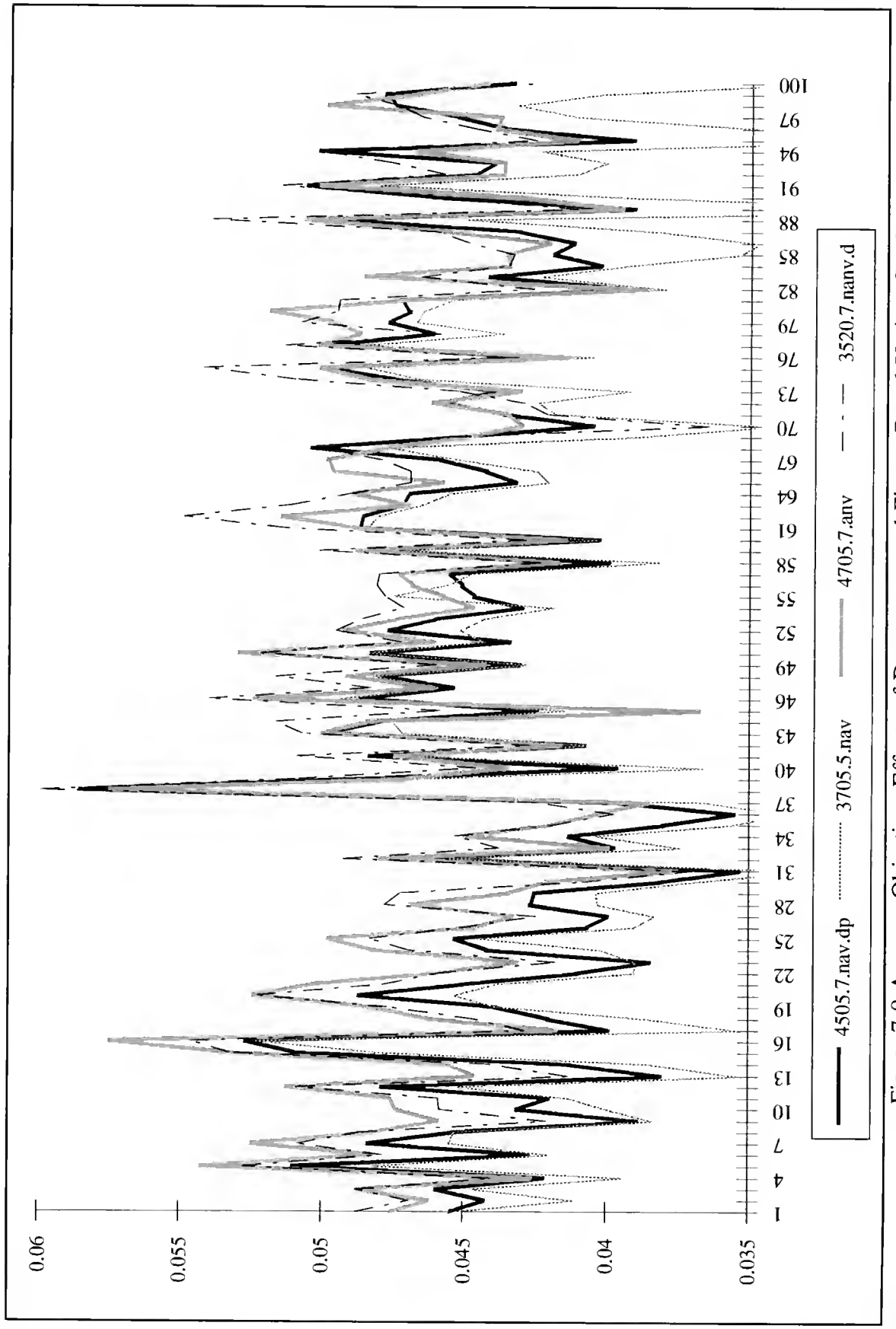

Figure 7-9 Average Objective--Effects of Degeneracy and Fitness Based Voting, Set 3

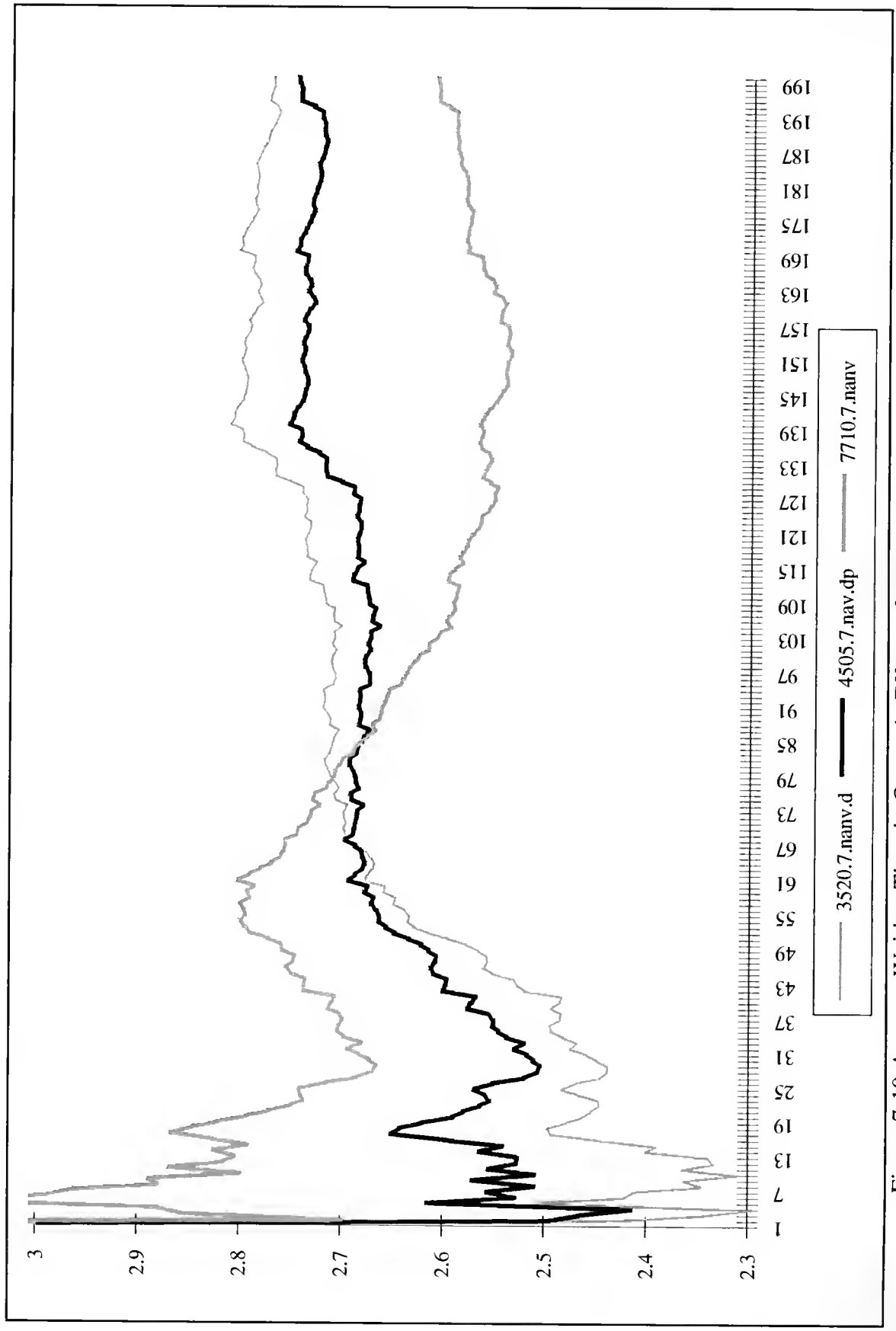

Figure 7-10 Average Waiting Time in Queue 1--Effects of Degeneracy and Fitness Based Voting, Set 3

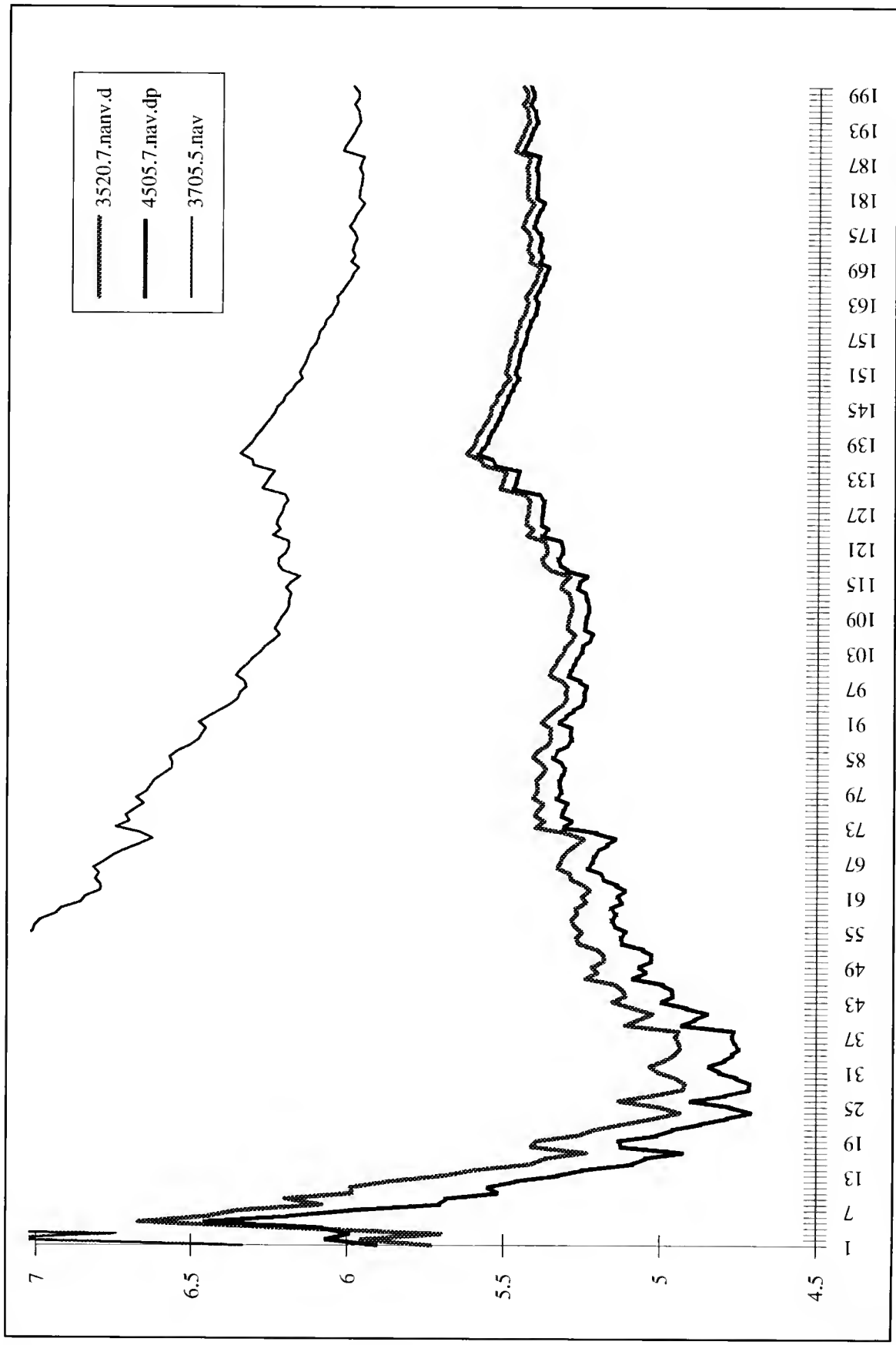

Figure 7-11 Average Waiting Time in Queue 2--Effects of Degeneracy and Fitness Based Voting, Set 3

CHAPTER 8
CONCLUSIONS AND FUTURE RESEARCH

8.1 Overview

In this research we have investigated applications of machine learning techniques to scheduling. We use a GA based learning paradigm because of reasons stated in Chapter 4. Hence, our research focuses on practical as well as theoretical issues of GA based learning with an adaptation of the technique to the scheduling problem. Our efforts focused on two aspects of the problem. While our practical focus was on developing a GA based dispatching system, theoretical research focused on developing a bound on the run time complexity of any GA algorithm for any problem.

Our practical and theoretical results are summarized below with an identification of possible areas of research emerging from our results.

8.2 Theoretical Research Results and Future Work

Results of Chapter 5 suggest that a random search is an upper bound on any GA for any given function. That is a GA is guaranteed to perform at least as good as a random search. It also suggests that increasing the mutation rate will help only up to a certain point and after this point the number of iterations required will increase. A population size of one suggest that in the absence of function information it is basically useless to have more than one population member. So using crossover does not help to

improve the performance of search in the presence of a function that does not convey much information. That is especially true for CSs where the fitness assignment function is not informative enough.

The lower bound obtained in Corollary 5.2 is not able to reflect most of the characteristics of GAs. First of all it does not take into consideration the F matrix, which reflects the function of interest. Secondly, it does not consider the crossover rate at all. It is a function of n, γ, and the confidence level, δ, only. One question that comes to mind is "are there any functions where the GAs act like pure random search algorithms?" The answer is yes. One trivial case is when the function of interest is a constant function. When this is the case there is no selection pressure and the iterates of a GA act like a "genetic drift" process. However, this is a trivial case. One challenging question is "are there any families of non-trivial functions such that the GA optimization is no better than a random search?"

Another challenge is to be able to use Theorem 3.2. to provide better bounds which incorporate information on the function to be optimized and the crossover rate. In the derivation of the spectral radius bound we have not considered any specific knowledge of the multinomial distribution inherent in Q. Perhaps a Chebychev type of inequality may yield better results than Corollary 5.2.

Theorem 5.2 provides an exact bound that assumes, with specified probability, that we have seen every population and hence every optimal solution. This is an overkill. We believe tighter bounds could be derived by establishing a way where all population members (not all populations) are seen at least once with specified probabilities. This

would not necessitate seeing all populations with specified probability and hence should provide a better bound.

8.3 Applied Research Results and Future Research

The preliminary study described in Chapter 6 had clearly demonstrated the ability of the system to improve its performance using GA based learning methods. The learning component was able to improve the simulation's performance by evolving a knowledge base. Even though the system was able to demonstrate the feasibility of the idea its restrictive rule language limited it to the use of dispatching rules only without considering the system or job related attributes. Below is the summary of our results based on an extension of work described in Chapter 6.

8.3.1 Conclusions

As a result of the simulations done by three different simulation setups, the system results are encouraging but still far from being applicable to real applications. The system is able to respond to changes in the environment as well as the changes in the evaluation functions. Out of three hundred simulations about fifty percent converged to a random knowledge base that is far from explaining the concept. Of those which performed well only a small percentage could retain the knowledge long enough to converge to the concept. But none of the simulations could reach the performance of the best rule, SPT. Even though the system outperformed some other simple dispatching heuristic there are

a couple of problems identified within the system some of which is discussed briefly in Chapter 7.

We have found that most of the rules do not persist even though they are performing well at the time. This may be due to our credit assignment scheme. Our credit assignment scheme is not biased enough to enforce the survival of these rules. This also may be due to the information passed back to the rules as fitness. Not all fitness functions are appropriate as we have discovered in Simulation Set One.

Our inference mechanism is not able to handle system and job related attributes appropriately. Since any conjunct is allowed to vote for any number of jobs, those that are too general are able to vote on every dispatching point for all the jobs. They get a fitness back no matter what the other rules decide since they vote for all the jobs. We observed that when such an attribute enters the knowledge base it is able multiply very rapidly and "degenerate" the knowledge base. When such a condition occurs the system behavior is very close to the RNDM dispatching rule. One way to solve this problem is develop more focused operators, and credit assignment scheme that favors focused conjuncts. We found out that our "Specialization" and "Generalization" operators are not very effective.

Our rules are basically high cardinality strings. It may be beneficial to map this coding to a binary string. One problem that we are facing now is that our crossover operator is not able to "explore" the search space effectively. Our only means of exploring new conjuncts is mutation. But with low mutation rate we are not able search

enough of the rule space, with high mutation rate the knowledge base does not stabilize and well performing rules can be lost.

As expected intermediate evaluation points significantly improved the system performance. The system is able to find the "fair share" of each dispatcher by evaluating them separately. However, this is not enough evidence to show that using intermediate points improves performance. An interesting question is "how can I measure the value added at each stage with respect to the performance measure?" If we can answer this question then implementing intermediate evaluation points becomes trivial.

One interesting phenomenon observed is that the knowledge base could retain high strength rules in the knowledge base without using them much. One easy solution to this is changing our static voting scheme to a fitness based voting scheme, where each rule assigns votes proportional to their fitness values. This will help not only to retain the well performing conjuncts and rules but will help them to dominate the current knowledge base until a better fit one is able to enter the knowledge base. Making this change improved the performance but it does not perform better consistently. Before we can stabilize the system parameters for a given environment it is hard to reach a conclusion.

8.3.2 Future Research and Modifications to the System

We have identified possible areas of research that may help to improve the performance of the system and contribute to the area.

The inference scheme and the credit assignment scheme will be changed such that focusing will be favored against extremely general rules.

To prevent the knowledge base from degenerating we will implement a "long-term memory." In this way we will be able to retain well performing rules and when a substantial decrease in the performance of the knowledge base is detected we will be able inject some of these back to boost the performance. A long-term memory will also serve as a filter where we can save those rules that are to be used in a non-learning mode.

We will implement different crossover operators under this coding and under a binary coding and compare the performances.

As discussed in the Section 8.3.1 finding a way to measure the value added by each activity is very important for correct credit assignment. We plan to investigate ways of measuring this in near future since credit assignment is the last and the most crucial point of this learning scheme.

Every learning episode the fitness of a conjunct has to be recalculated before a learning event can take place. This keeps the learning episode quite long, since we want to make sure that we estimate the true fitness values of the rules. Instead we could investigate ways of carrying part of the existing fitness values to the next generation based on average schema fitnesses. Goldberg (1989a), (1989b) discusses the use of Walsh coefficients to calculate the average schema fitnesses which is very hard to compute directly. Such a scheme can be utilized for computational efficiency.

We do not yet know if an optimal set of parameters exist for any given learning problem. It would be a big contribution to find out what parameter ranges yield the best performance. This would make this approach as general as it could be and reduce the computational burden on searching for "the best" parameter settings.

REFERENCES

Adachi, T., J. J. Talavage , and C.L. Moodie, "A Rule Based Control Method for a Multi-loop Production System," *Artificial Intelligence in Engineering*, 1989, 4, No. 3, pp. 115-125.

Alpar, P. and K.N. Srikanth, "A Comparison of Analytic and Knowledge-Based Approaches to Closed-Shop Scheduling," *Annals of Operations Research*, 1989a, 17, pp. 347-362.

Alpar, P. and K.N. Srikanth, "Closed-Shop Scheduling with Expert System Techniques," Knowledge-based Systems in Manufacturing, 1989b, Ed. A. Kusiak, New York, NY: Taylor and Francis, pp. 247-264.

Angluin, D. and P. Laird, "Learning From Noisy Examples," *Machine Learning*, 2, 1988, pp. 343-370.

Arff, S., and G. Hasle, "Synthesis of Schedules Using Heuristic, Constraint-Guided Search," *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, Hilton Head, S.C., May 1990, pp. 111-124.

Baker, K. R., Introduction to Sequencing and Scheduling, 1974, New York, NY: John Wiley & Sons Inc.

Baker, Thomas E., "Integrating AI/OR/Database Technologies for Production Planning and Scheduling," *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, Hilton Head, S.C., May 1990, pp. 101-110.

Barr A., and E.A. Feigenbaum, The Handbook of Artificial Intelligence, Reading, MA: Addison-Wesley Publishing Company, Inc., Vol. III, 1989.

Bazaraa M. S., C. M. Shetty, Nonlinear Programming Theory and Algorithms, 1979, New York: Wiley.

Bel, G., E. Bensana, D. Dubois, J. Erschier, P. and Esquirol, "A Knowledge-Based Approach to Industrial Job-Shop Scheduling," Knowledge-Based Systems in Manufacturing, Ed. A. Kusiak, 1989, Taylor & Francis, pp. 207-246.

Ben-Arieh, D., "Knowledge-Based Control System for Automated Production and Assembly," Modelling and Design of Flexible Manufacturing Systems, Ed. A. Kusiak, New York, NY: Elsevier, 1986, pp. 347-368.

Bensana, E., M. Correge, G. Bel, and D. Dubois, "An Expert-System Approach to Industrial Job-Shop Scheduling," *Proceedings of 1986 IEEE International Conference on Robotics and Automation*, San Francisco, April 7-10, pp 1645-1650.

Bhaskaran, K. and M. Pinedo, "Dispatching," Handbook of Industrial Engineering, Ed. Gavriel Salvendy, New York, NY: Wiley, 1992.

Blackstone, J. H., D.T. Phillips, and G.L. Hogg, "The State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations," *International Journal of Production Research*, 1982, 20, No. 1, pp. 27-45.

Blessing, J.A., and B.A. Watford, "INFMSS, An Intelligent FMS Scheduling System," *World Productivity Forum and 1987 Annual International Industrial Engineering Conference Proceedings*, Norcross, GA, 1987, pp. 82-88.

Board, R. and L .Pitt, "Semi-Supervised Learning," *Machine Learning*, 4, 1989, pp. 41-65.

Braun, H. and J. S. Chandler, "Predicting Stock Market Behavior Through Rule Induction: An Application of the Learning-from-Example Approach," *Decision Sciences*, 1987, 18, pp. 415-429.

Brown, R., "Knowledge-Based Scheduling and Resource Allocation in the CAMPS Architecture," Intelligent Manufacturing, *Proceedings of the First International Conference on Expert Systems and the Leading Edge in Production Planning and Control*, 1988, M.O. Oliff, Benjamin/Cummings, pp. 165-186.

Carbonell J. G., R. S. Michalski and T. M. Mitchell, "Machine Learning: A Historical and Methodological Analysis," *The AI Magazine*, Fall 1983, 4, No. 3, pp. 69-77.

Carter, C. and J. Cotlett, "Assessing Credit Card Applications Using Machine Learning," *IEEE Expert*, Fall 1987, pp. 71-79.

Chang, F.C., "A Knowledge-Based Real-Time Decision Support System for Job Shop Scheduling at the Shop Floor Level," Ph.D. Dissertation, Department of Industrial and Systems Engineering, Ohio State University, May 1985.

Chryssolouris, G., K. Wright, J. Pierce and W. Cobb, "Manufacturing Systems Operation: Dispatch Rules Versus Intelligent Control," *Robotics and Computer-Integrated Manufacturing*, 1988, 4, No. 3/4, pp. 531-544.

Chryssolouris, G., J.E. Pierce and W. Cobb, "A Decision-Making Approach to the Operation of FMS," *Proceedings of the Third ORSA/TIMS Conference on Flexible Manufacturing Systems*: 1989, Operations Research Models and Applications, pp. 355-360.

Chryssolouris G., Lee M. and Domroese M., "The Use of Neural Networks in Determining Operational Policies for Manufacturing Systems," *Journal of Manufacturing Systems*, 1990, Vol. 10, No. 2, pp.166-175.

Clancy, D.P. and S. Mohan, "RBD - Rule Based Job Dispatching Software for Implementing Production Plans," *First International Conference on Expert Planning Systems*, Metropole Hotel, Brighton, June 1990, pp. 100-103.

Clark, P. and T. Niblett, "The CN2 Induction Algorithm," *Machine Learning*, 3, 1989, pp. 261-283.

Cleveland G. A., and S. F. Smith, "Using Genetic Algorithms to Schedule Flow Shop Releases," *Proceedings of the Third International Conference on Genetic Algorithms*, Schaffer, J. E. editor, George Mason University, 1989, pp. 160-169.

Cohen P. R., and E. A., Feigenbaum, The Handbook of Artificial Intelligence, volume III, Reading, MA: Addison-Wesley, 1982.

Conway, R., and W. Maxwell, "Low-Level Interactive Scheduling," *Symposium on Real Time Optimization in Automated Manufacturing Facilities*, National Bureau of Standards, Gaithersburg MD, April 1986, pp. 99-107.

Czigler, M.H., and Whitaker, C.R., "A Hybrid Algorithmic and Knowledge-Based Implementation for Workcenter-Based Production Scheduling," *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, Hilton Head, S.C., May 1990, pp. 145-155.

Davis R., "TEIRESIAS: Applications of Meta-Knowledge," Knowledge Based Systems in Artificial Intelligence, Eds. R. Davis and D. Lenat, New York: McGraw-Hill, 1982.

Davis, T., "Toward an Extrapolation of The Simmulated Annealing Convergence Theory Onto The Simple Genetic Algorithm," Dissertation, University of Florida, 1991.

Day, J. E., and M. P. Hottenstein, "Review of Sequencing Research," *Naval Research Logistic Quarterly*, 17, 1970, pp. 11-39.

De, S., "A Knowledge-based Approach to Scheduling in an FMS," *Annals of Operations Research*, 1988, 12, pp. 109-134.

De, S., and A. Lee, "A Knowledge-Based System for Flexible Assembly Scheduling," *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, Hilton Head, S.C., May 1990, pp. 156-167.

Dejong, G., and R. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning*, 1, 1986, pp. 145-176.

Erschler, J. and P. Esquirol, "Decision-Aid in Job Shop Scheduling: A Knowledge Based Approach," *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 7-10, 1986, pp. 1651-1656.

Erschler, J. and F. Roubellat, "An Approach to Solve Workshop Real Time Scheduling Problems," Advanced Information Technologies for Industrial Material Flow Systems, Ed. S.Y. Nof and C.L. Moodie, Berlin: Springer-Verlag, 1989, pp. 651-679.

Farhoodi, F., "A Knowledge-Based Approach to Dynamic Job-Shop Scheduling," *International Journal of Computer Integrated Manufacturing*, 1990, 3, No. 2, pp. 84-95.

Flann, N. S., and T. G. Dietterich, "A Study of Explanation-Based Methods for Inductive Learning," *Machine Learning*, 4, 1989, pp. 1878-226.

Fox, B.R., and K.G. Kempf, "Reasoning About Opportunistic Schedules," *IEEE*, 1987, pp. 1876-1882.

Fox, M.S., "Constraint Directed Search: A Case Study of Job-Shop Scheduling," Ph.D. Th., Carnegie-Mellon University Pittsburgh, PA, October 1983, Technical Report CMU-RI-TR-83-22.

Fox, M. S., "Constraint-Guided Scheduling--A Short History of Research at CMU," *Computers in Industry*, 1990, 14, pp. 79-88.

Fox, M.S. and N. Sadeh, "Why is Scheduling Difficult? A CSP Perspective," *Proceedings of the 9th European Conference on Artificial Intelligence*, Stockholm, Sweden, August 6-10, 1990, pp. 754-767.

Fox, M. S., N. Sadeh, and C. Baykan, "Constrained Heuristic Search," *Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, August 20-25, 1989, pp. 309-315.

Fox, M.S., and S.F. Smith, "The Role of Intelligent Reactive Processing In Production Management," *Proceedings of CAM-I's 13th Annual Meeting and Technical Conference*, Clearwater Beach, FL, November 13-15, 1984a, pp. 6.13-6.17.

Fox, M.S., and S.F. Smith, "ISIS-A Knowledge-Based System for Factory Scheduling," *Expert Systems*, 1984b, 1, No. 1, pp. 25-44.

Fox, M. S. and K.P. Sycara, "Overview of CORTES: A Constraint Based Approach to Production Planning, Scheduling and Control," *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, Hilton Head, May, 1990, pp. 17-25.

Frey, P. W. and D. J. Slate, "Letter Recognition using Holland-Style Adaptive Classifiers," *Machine Learning*, 1991, 6, No. 2, pp. 161-182.

Gary M. R., and D. S. Johnson, Computers and Intractability A Guide to the Theory of NP-Completeness, New York, NY: W. H. Freeman and Company, 1979.

Goldberg, D. E., Genetic Algorithms in Search, Optimization and Machine Learning, 1989a, Reading, MA: Addison Wesley.

Goldberg D. E., "Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction," *Complex Systems*, 3, 1989b, pp. 129-152.

Goldberg D. E., "Genetic Algorithms and Walsh Functions: Part II, Deception and Its Analysis," *Complex Systems*, 3, 1989c, pp. 153-171.

Goldberg, D. E., "Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding," *Machine Learning*, 5, 1990, pp. 407-425.

Goldberg, D. E., B. Korb, and K. Deb, "Messy Genetic Algorithms: Motivation, Analysis, and First Results," *Complex Systems*, 3, 1989, pp. 493-530.

Goldberg, D. E., B. Korb, and K. Deb, "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale," *Complex Systems*, 4, 1990, pp. 415-444.

Goldberg D. E., and P. Segrest, "Finite Markov Chain Analysis of Genetic Algorithms," *Second International Conference on Genetic Algorithms*, 1987, Cambridge, MA, pp. 2-7.

Grant, T.J., "Lessons for O.R. From A.I.: A Scheduling Case Study," *Journal of Operational Research Society*, 1986, 37, No. 1, pp. 41-57.

Graves, S.C., "A Review of Production Scheduling," *Operations Research*, April, 1981, 2914, pp. 646-675.

Greene, D. P. and S. F. Smith, "COGIN: Symbolic Induction with Genetic Algorithms," 1992, working paper, Graduate School of Industrial Administration-School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

Grefenstette, J. J., "Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms," *Machine Learning*, 3, 1988, pp. 225-245.

Grefenstette, J. J., "Incremental Learning of Control Strategies with Genetic Algorithms," *Proceedings of the Sixth International Conference on Machine Learning*, Cornell University, Ithaca, N.Y., 1989, Segre A. M. Editor, San Mateo, CA: Morgan Kaufmann Publishers Inc., pp. 340-344.

Haussler, D., "Applying Valiant's Learning Framework to AI Concept Learning Problems," Paper UCSC-CRL-87-11, Baskin Center for Computer Engineering and Information Sciences, University of California, Santa Cruz, 1987.

Haussler D., "Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework," *Artificial Intelligence*, 36, 1988, pp. 177-221.

Haussler D., "Learning Conjunctive Concepts in Structural Domains," *Machine Learning*, 4, 1989, pp. 7-40.

Hecht-Nielsen, R., Neurocomputing, 1990, Addison-Wesley Publishing Company Inc.

Hilliard, M. R., G. E. Liepins, G., M. Palmer, M. Morrow, and J. Richardson, "A Classifier-Based System for Discovering Scheduling Heuristics," *Proceedings of the Second Internatioanl Conference on Genetic Algorithms and Their Applications*, Cambridge, Massachusetts, 1987, pp. 231-235.

Hilliard, M. R., G. Liepins, G. Rangarajan, and M. Palmer, "Learning Decision Rules for Scheduling Problems: A Classifier Hybrid Approach," *Proceedings of the Sixth Internatioanl Conference on Machine Learning*, Cornell University, Ithaca, N.Y., 1989, Segre A. M. Editor, San Mateo, CA: Morgan Kaufmann Publishers Inc., pp. 188-190.

Husbands P., F. Mill, and S. Warrington, "Genetic Algorithms, Production Plan Optimization and Scheduling," Parallel Problem Soving from Nature, eds. H. Schwefel and R. Manner, Lecture Notes in Computer Science, No. 496, New York, NY: Springer Verlag, 1991, pp. 80-84.

Jain, S., and D. Osterfeld, "Expert Simulation for On-line Scheduling," *Winter Simulation Conference*, Washington D.C., 1989, pp. 930-935.

Kak, A.C., Boyer, K. L., Chen, C.H., Safranek, R. J., and Yang, H. S., "A Knowledge-Based Robotic Assembly Call," *IEEE Expert*, Spring 1986, pp. 63-83.

Kerr, R.M., and R.V. Ebsary, "Implementation of an Expert System for Production Scheduling," *European Journal of Operational Research*, 1988, 33, pp. 17-29.

Koehler, G. J., "A Proof of the Vose-Liepins Conjecture," working paper, 1992, Decision and Information Sciences, BUS 351, University of Florida, Gainesville, FL 32611.

Koton, P., "SMARTplan: A Case-Based Resource Allocation and Scheduling System," *Proceedings of the DARPA Workshop on Case-Based Reasoning*, Pensacola Beach FL, 1989, Ed. Kaufmann M., Hammond K., pp. 285-289.

Kusiak A., and M. Chen, "Expert Systems For Planning and Scheduling Manufacturing Systems," *European Journal of Operations Research*, 1988, 34, pp. 113-130.

Lageweg, B. J., E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Computer Aided Complexity Classification of Deterministic Scheduling Problems," 1981, Mathematisch Centrum, Department of Operations Research, Amsterdam, The Netherlands.

Lamatsch, A., M. Morlock, K. Neumann, and T. Rubach, "SCHEDULE-An Expert-like System for Machine Scheduling," *Annals of Operations Research*, 1988, 16, pp. 425-438.

Lawler E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, "Sequencing and Scheduling: Algorithms and Complexity," Report BS-R8909, 1985, Center for Mathematics and Computer Science, Amsterdam, The Netherlands.

Lawrence, S.R., and T.E. Morton, "Patriarch: Hierarchical Production Scheduling," *Symposium on Real Time Optimization in Automated Manufacturing Facilities*, National Bureau of Standards, Gaithersburg MD, January 1986, pp. 87-97.

Lecocq, P., and T. Guiot, "An Expert Systems Application to Increase the Flexibility and Efficacy of Real-time FMS Controllers," Expert Systems and Intelligent Manufacturing, 1988, Ed. M. Oliff, Elsevier Science Publishing Company, pp. 249-264.

Lee, J.K. and M.S. Suh, "PAMS: A Domain-Specific Knowledge-Based Parallel Machine Scheduling System," *Expert Systems*, 1988, 5, No. 3, pp. 198-214.

Lee, T., "A Logical Framework using Prolog for Expert Scheduler in Manufacturing," *Proceedings of the IXth ICPR*, Cincinnati, August, 1987, pp. 1128-1134.

Liepins, G. E., M. R. Hilliard, M. Palmer, and G. Rangarajan, "Credit Assignment and Discovery in Classifier Systems," *Internatioanl Journal of Intelligent Systems*, Vol. 6, 1991, pp. 55-69.

Liepins, G. E., and M. D. Vose, "Representational Issues in Genetic Optimization," *Journal of Expert Theory Artificial Intelligence*, 2, 1990, pp. 101-115.

Liepins, G. E., and M. D. Vose,"Deceptiveness and Genetic Algorithm Dynamics," Foundations of Genetic Algorithms, Ed. G. Rawlins, 1991a, San Mateo, CA: Morgan Kaufman Publishers.

Liepins, G. E., and M. D. Vose, "Polynomials, Basis Sets, and Deceptiveness in Genetic Algorithms," *Complex Systems*, 5, 1991b, No.1, pp. 45-61.

Liepins, G. E., and M. D. Vose, "Characterizing Crossover in Genetic Algorithms," *Annals of Mathematics and AI*, 5, 1992, pp. 27-34.

Liepins G. E., L. A. Wang, "Classifier System Learning of Boolean Concepts," *Proceedings of the Fourth International Conference on Genetic Algorithms*, University of California, San Diego, July 13-16, 1991, pp. 318-323.

Martinez, J., P.R. Muro, M. Silva, S.F. Smith, and J.L. Villarroel, "Merging Artificial Intelligence Techniques and Petri Nets for Real Time Scheduling and Control of Production Systems," AI and Expert Systems in Scientific Computing, of the IMACS Transactions on Scientific Computing-88 series, 1988.

May, J. H., L.G. Vargas, R.De', S.A. Slotnick, T.E. Morton, D.W. Pentico and G.J. Tatar, "Multex: An Integrated AI/OR System For Factory Scheduling," *Proceedings of the Northeast DSI Conference*, April 1991, Pittsburgh.

Mc Kay, K.N., J.A. Buzacott, N. Charness, and F.R. Safayeni, "The Scheduler's Predictive Expertise-An Inter-disciplinary Perspective," Artificial Intelligence in Operational Research, Eds. G. I. Doukidis, R. J. Paul, Basingstoke, England: Macmillan Publishing, 1992.

Messier, W. F., and J. V. Hansen, "Inducing Rules for Expert System Development: An Example Using Default and Bankruptcy Data," *Management Science*, 34, No. 12, December 1988, pp. 1403-1415.

Michalski R. S., "A Theory and Methodology of Inductive Learning," *Artificial Intelligence*, 20, 1983, pp. 111-161.

Miller, R.K., E. Lufg, and T.C. Walker, Artificial Intelligence Applications in Manufacturing, 1988, pp. 166-177.

Minc, H., Nonnegative Matrices, 1988, New York, NY: John Wiley & Sons.

Mingers, J., "An Empirical Comparison of Selection Measures for Decision-Tree Induction," *Machine Learning*, 3, 1989a, pp. 319-342.

Mingers, J., "An Empirical Comparison of Pruning Methods for Decision Tree Induction," *Machine Learning*, 4, 1989b, pp. 227-243.

Mitchell, T. M., R. M. Keller, and S. T. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning*, 1, 1986, pp. 0.

Mohan, S., and D. Clancy, "SIS-Rule Based Software for Automating Job Dispatch on the Factory Floor," *Proceedings from the Third International Conference on Expert Systems and the Leading Edge in Production Planning and Control*, 1989.

Muller, (-Malek) H., D. De Samblanckx, and D. Matthys, "The Expert System Approach and the Flexibility-Complexity Problem in Scheduling Production Systems," *International Journal of Production Research*, 1987, 25, No. 11, pp. 1659-1670.

Nakasuka, S., T. Yoshida, "Dynamic SCheduling System Utilizing Machine Learning as a Knowledge Acquisition Rule," *International Journal of Production Research,* 1992, 30, No. 2, pp. 411-431.

Natarajan, B. K., Machine Learning A Theoretical Approach, 1991, San Mateo, CA: Morgan Kaufmann Publishers Inc.

Newman, P.A. and K.G. Kempf, "Opportunistic Scheduling For Robotic Machine Tending," The *2nd Conference on Artificial Intelligence Applications*, Miami Beach, FL, December 11-13, 1985, pp. 168-173.

Niew, B.C., B.S. Lim, and N.C. Ho, "Knowledge Based Master Production Scheduler," *First International Conference on Expert Planning Systems*, Brighton, UK, June 27-29 1990, pp. 88-93.

Nix, A. E., and M. D. Vose, "Modeling Genetic Algorithms with Markov Chains," *Annals of Mathematics and Artificial Intelligence*, 5, 1992, pp. 79-88.

O'Grady, P., and K.H. Lee, "An Intelligent Cell Control System for Automated Manufacturing," *International Journal of Production Research*, 1988, 26, No. 5, pp. 845-861.

O'Keefe, R., "Simulation and Expert Systems-A Taxonomy and Some Examples," *Simulation*, 1986, 46, No. 1, pp. 10-16.

O'Rorke P., "LT Revisited: Explanation-Based Learning and the Logic of Principia Mathematica," *Machine Learning*, 4, 1989, pp. 117-159.

Ow, P.S., "Experiments in Knowledge-based Scheduling," Technical Report, 8, April 1986.

Ow, P.S., S.F. Smith, and A. Thriez, "Reactive Plan Revision," *AAAI*, 1988.

Ow, P.S., S.F. Smith and R. Howie, "A Cooperative Scheduling System," 1988 Expert Systems and Intelligent Manufacturing, Ed. M.D. Ol. FF, North-Holland, pp. 70-89.

Park, S.C., N. Raman and M.J. Shaw, "Heuristic Learning in Pattern-Directed Scheduling," 1989, *Proceedings of the Third ORSA/TIMS Conference on Flexible Manufacturing Systems*, Elsevier Science, Amsterdam.

Parker, C.S., Management Information Systems: Strategy and Action, 1989, MacGraw Hill.

Parunak, H.V.D., "Distributed AI Systems," Ed. A. Kusiak, Artificial Intelligence: Implications for Computer Integrated Manufacturing, IFS Publications, Kempston, U.K., and Springer-Verlag, New York, pp. 225-251, 1988.

Piramuthu, S., S-C. Park, N. Raman, and M.J. Shaw, "Integration of Simulation Modeling and Inductive Learning in an Adaptive Decision Support System," Model Management Systems, eds. Bonczelc and A.B. Whinston, IEEE Society Press, 1991.

Powner, E.T. and D.H. Walburn, "A Knowledge Based Scheduler," *First International Conference on Expert Planning Systems Brighton UK*, June 27-29 1990, pp. 82-87.

Quinlan, J. R., "Induction of Decision Trees," *Machine Learning*, 1, 1986, pp. 81-106.

Raghavan, V., "An Expert System Framework for the Management of Due Dates in Flexible Manufacturing Systems," Expert Systems and Intelligent Manufacturing, 1988, Ed. M.D. Oliff, Elseview Science Publishing Company, pp. 235-247.

Ready, C.M., W.H. Simmonds, and J.C. Taunton, "A Knowledge Based Planning and Scheduling Toolkit for the Process Industries," *First International Conference on Expert Planning Systems Brighton UK*, June 27-29, 1990, pp. 110-113.

Reinschmidt, K.F., J.H. Slate, and G.A. Finn, "Expert Systems for Plant Scheduling using Linear Programming," *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, Hilton Head S.C., May, 1990, pp. 198-211.

Rivest, R. L., "Learning Decision Lists," *Machine Learning*, 2, 1987, pp. 229-246.

Rodammer, F.A., and K.P. White, "A Recent Survey of Production Scheduling," *IEEE Transactions on Systems, Man and Cybernetics*, 1988, 18, No. 6, pp. 841-851.

Sadeh, N., and M.S. Fox, "Variable and Value Ordering Heuristics for Activity-Based Job-Shop Scheduling," *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, Hilton Head, S.C., May, 1990, pp. 26-36.

Sadeh, N. and M.S. Fox, "Preference Propagation in Temporal/Capacity Constraint Graphics," paper no. CMU-RI-TR-89-2, Carnegie Mellon University, Pittsburgh, PA, 1989.

Sarin, S.C., and R. Salgame, "A Knowledge-Based System Approach to Dynamic Scheduling," Knowledge-Based Systems in Manufacturing, Ed. A. Kusiak, 1989, Taylor and Francis, pp. 173-206.

Savell, D.V., R.A. Perez, and S.W. Koh, "Scheduling Semiconductor Wafer Production: An Expert System Implementation," *IEEE Expert*, Fall 1989, pp 9-15.

Seneta, E., Non-Negative Matrices, 1973, London: Gergo Allen & Unwin Ltd.

Shaw, M.J., "A Two Level Planning and Scheduling Approach for Computer Integrated Manufacturing," *Symposium on Real Time Optimization in Automated Manufacturing Facilities*, National Bureau of Standards, Gaithersburg, MD, January, 1986, pp. 187-194.

Shaw, M.J., "FMS Scheduling as Cooperative Problem Solving," *Operations Research*, 1988a, 17, pp. 323-346.

Shaw, M.J., "Dynamic Scheduling in Cellular Manufacturing Systems: A Framework for Networked Decision Making," *Journal of Manufacturing Systems*, 1988b, 7, No. 2, pp. 83-94.

Shaw, M.J. "Knowledge-based Scheduling in Flexible Manufacturing System: An Integration of Pattern-directed Inference and Heuristic Search," *International Journal of Production Research*, 1988c, 26, No. 5, pp. 821-844.

Shaw, M.J., N. Raman and S.C. Park, "Intelligent Scheduling with Machine Learning Capabilities: The Induction of Scheduling Knowledge," Technical Report AI-DSS-91-01, Beckman Institute, University of Illinois, Urbana-Champaign, 1991.

Shaw, M.J. and A.B. Whinston, "Application of Artificial Intelligence to Planning and Scheduling in Flexible Manufacturing," Flexible Manufacturing Systems: Methods and Studies, Ed. A. Kusiak, Amsterdam, Netherlands: North Holland, pp. 223-242, 1986.

Shaw, M.J. and A.B. Whinston, "Task Bidding and Distributed Planning in Flexible Manufacturing," *Second Conference on Artificial Intelligence Applications*, IEEE Computer Society, Miami Beach, FL, December, 1985, pp. 184-189.

Shen, S. and Y. Chang, "An AI Approach to Schedule Generation in Flexible Manufacturing," Eds. K.E. Stecke and R. Suri, Flexible Manufacturing Systems: Operations Research Models and Applications, Elsevier, New York, pp. 581-592, 1986.

Simon, H.A., "Two Heads Are Better Than One: The Collaboration between AI and OR," *Interfaces*, 1987, 17, No. 4, pp. 8-15.

Smith, S.F., "A Constraint-Based Framework for Reactive Management of Factory Schedules," Intelligent Manufacturing, *Proceedings of the First International Conference on Expert Systems and the Leading Edge in Production Planning and Control*, 1988a, M.D. Oliff, Benjamin/Cummings, pp. 113-130.

Smith, S.F., "Knowledge-Based Scheduling Systems," *Proceedings from the First International Conference on Expert Systems and the Leading Edge in Production Planning and Control*, 1988b, M. Oliff, Benjamin/Cummings, pp. 381-383.

Smith, S.F., P.S. Ow, and J-Y. Potvin, "OPIS: An Opportunistic Factory Scheduling System," *The Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, IEA/AIE 90, Charleston, July 15-18, 1990.

Steffen, M.S., "A Survey of Artificial Intelligence-Based Scheduling Systems," *Proceedings Fall Industrial Engineering Conference*, December 7-10, 1986, pp. 395-405.

Steffen, M.S., and T.J. Greene, "A Prototype System for Scheduling Parallel Processors Using Artificial Intelligence Methods," 1986 *Annual International Industrial Engineering Conference Proceedings*, 1986a, pp. 156-164.

Steffen, M.S., and T.J. Greene, "Hierarchies of Sub-Periods in Constraint-Directed Scheduling," *Symposium on Real Time Optimization in Automated Manufacturing Facilities*, National Bureau of Standards, Gaithersburg MD, January, 1986b, pp. 167-183

Steffen, M.S., and T.J. Greene, "Automating the Scheduling of Parallel Machines," Smart Manufacturing with Artificial Intelligence, Ed. J. Krakauer, Dearborn, MI: Computer and Automated Systems Association of SME, 1987, pp. 119-134.

Subramanyam, S. and R.G. Askin, "An Expert System Approach to Scheduling in Flexible Manufacturing Systems," Flexible Manufacturing Systems: Methods and Studies, Ed. A. Kusiak, Amsterdam, Netherlands: North Holland, pp. 243-256, 1986.

Sullivan, G., and K. Fordyce, "IBM Burlington's Logistics Management System," *Interfaces*, 1990, 20, No. 1, pp. 43-64.

Tanimoto, S. L., The Elements of Artificial Intelligence, 1990, New York, NY: W. H. Freeman and Company.

Thesen A., and L. Lei, "An 'Expert' System for Scheduling Robot in a Flexible Electroplating System with Dynamically Changing Work Loads," *Proceedings of the Second ORSA/TIMS Conference on Flexible Manufacturing Systems: Operations Research Models and Applications*, Ed. K.E. Stecke, Amsterdam, The Netherlands: Elsevier Science Publishers B.V., 1986, pp. 555-566.

Valiant, L. G., "A Theory of the Learnable," *Communications of the ACM*, Vol. 27, No. 11, November 1984, pp. 1134-1142.

Vepsalainen A. P. J., Morton T. E., "Priority Rules for Job Shops with Weighted Tardiness Costs," *Management Science*, Vol. 33, No. 8, August 1987, pp. 1035-1047.

Vose,M. D., and G. E. Liepins, "Punctuated Equilibria in Genetic Search," *Complex Systems*, 5, 1991a, no. 1, 31-44.

Vose, M. D., and G. E. Liepins, "Schemata Disruption," to appear in *Fourth International Conference on Genetic Algorithms*, University of California, San Diego, July 13-16, 1991b, pp. 237-242.

Warwick, A.M., and H.M.J. Walters, "A Rule Based Planning and Scheduling System For Manufacturing Industries," *First International Conference on Expert Planning Systems*, Brighton UK, June 27-29, 1990, pp. 104-109.

Whitley, D., T. Starkweather, F'A. Fuquay, "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator," *Proceedings of the Third International Conference on Genetic Algorithms*, Schaffer, J. E. editor, George Mason University, 1989, pp.133-140.

Wilson, S. W., "Classifier Systems and the Animat Problem," *Machine Learning*, 2, 1987, pp. 199-228.

Woodhead, R., Z. Dobolyi, and A.D. De Pennington, "Process Planning as an Application for Expert Systems Technology," *PED-Vol. 21, Integrated and Intelligent Manufacturing*, Eds. C.R. Liu, and T.C. Chang, ASME, WAM, Anaheim, CA, December 7-12, 1986, pp. 143-156.

Worral, B.M., and E.L. MacDonald, "A Production Scheduling System Incorporating Resource Constraints and Order Priorities," *Proceedings of the IXth ICPR*, Cincinnati, August, 1987, pp. 2732-2738.

Yang, D.L., and H-B. Jiang, "A Production Planning Expert System in Manufacturing," *First International Conference on Expert Planning Systems*, June 27-29, 1990, pp. 28-32.

Yih, Y., "Trace Driven Knowledge Acquisition (TDKA) for Rule Based Real Time Scheduling Systems," *Journal of Intelligent Manufacturing*, 1990, 1, pp. 217-230.

Yih, Y., and A. Thesen, "Semi-Markov Decision Models for Real-Time Scheduling," *International Journal of Production Research*, 1991, 29, No. 11, pp. 2331-2346.

Yih, Y., "Learning Real-Time Scheduling Rules from Optimal Policy of Semi-Markov Decision Processes," *International Journal of Computer Integrated Manufacturing*, 1992, 5, No. 3, 171-181.

Yih, Y., T.-P. Liang, and H. Moskowitz, "Robot Scheduling in a Circuit Board Production Line - A Hybrid OR/ANN Approach," working paper, 1992, School of Industrial Engineering, Purdue University, West Lafayette, IN 47907.

Zangwill, W. I., Nonlinear Programming: A unified Approach, 1969, Englewood Cliffs, NJ: Prentice-Hall Inc.

Zhou H. H., "CSM: A Computational Model of Cumulative Learning," *Machine Learning*, 5, 1990, pp. 383-406.

BIOGRAPHICAL SKETCH

Haldun Aytug received his Bachelor of Science degree in industrial engineering, in July 1988 from Boğazici University, Istanbul/Turkey. He started his graduate studies in decision and information sciences at the University of Florida in August 1989.

The author has been a teaching assistant for the Introduction to Management Information Systems course for several semesters and has taught the Managerial Operations Analysis III course in Spring 1993. He has participated in a project with IBM Corporation on the applicability of genetic algorithm based learning on scheduling. His research concentrates on machine learning and artificial intelligence applications in business.

He is a member of Beta Gamma Sigma and Alpha Iota Delta honorary societies and The Institute of Management Science.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Gary J. Koehler, Chairman
Professor of Decision and
Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Chung Yee Lee
Associate Professor of
Industrial and Systems
Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Richard A. Elnicki
Professor of Decision and
Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Anthal Majthay
Associate Professor of
Decision and Information
Sciences

This dissertation was submitted to the Graduate Faculty of the Department of Decision and Information Sciences in the College of Business Administration and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 1993

Dean, Graduate School